# MC920 – Trabalho 5

Rafael Sartori M. Santos, 186154

21 de novembro de 2019

## 1 Introdução

O objetivo do trabalho é avaliar o procedimento PCA (*Principal Component Analysis*) como efetivo método de compressão de imagens através da redução de componentes menos significativos.

Faremos isso através de um programa Python que utiliza as bibliotecas *OpenCV*, *NumPy* e padrão.

## 2 Metodologia

Analisaremos a compressão feita com a redução de componentes dados pelo PCA nas imagens dadas pelo professor no enunciado do trabalho e em uma imagem de grande dimensão (comparada às outras) capturada por um celular comum moderno.

Todas as imagens utilizadas estavam no formato `PNG`, como especificado no enunciado, para não serem afetadas pela compressão com perdas de outros métodos. Durante o desenvolvimento do projeto, sobreescrevemos as imagens originais utilizando a biblioteca *OpenCV* (a mesma que utilizaremos para salvar as imagens alteradas), de forma a minimizar os efeitos das diferentes implementações do formato `PNG`.

Começamos o programa descrevendo diversas opções de entrada para o usuário: qual imagem será utilizada, o parâmetro $k$ para redução de componentes, onde salvaremos a imagem final, o relatório (em texto) de análise da compressão e ainda se utilizaremos a implementação para SVD (*Singular Value Decomposition*, uma das maneiras de se fazer PCA) já feita de *NumPy* ou, caso contrário, se utilizaremos a nossa implementação.

Utilizando essas opções, abrimos a imagem através de *OpenCV*, isolando as camadas de cores para serem tratadas individualmente e as transformando em vetor de pontos flutuantes.

### 2.1 Utilizando SVD de *NumPy*

Na implementação utilizando SVD de *NumPy*, fazemos a decomposição e, posteriormente, pegamos apenas os $k$ primeiros componentes mais significativos para refazer a imagem. Isso pode ser visto na eq. (1), considerando $g$ a imagem final e $f$, a inicial.

$$U, D, V^T = \texttt{svd}(f) \tag{1a}$$

$$g = U_{[:,0:k]} D_{[0:k]} V^T_{[0:k,:]} \tag{1b}$$

Na eq. (1a), *NumPy* já ordena decrescentemente o resultado de SVD, o que é mais amigável as aplicações que utilizam isso para fazerem PCA. Desta forma, é bem rápido e fácil de implementar um algoritmo de compressão que utiliza PCA, mas não vemos com detalhes as operações realizadas nem a interpretação não usual da imagem como componentes.

### 2.2 Implementação própria

Já na implementação própria de PCA, ao contrário, fazemos a compressão em detalhes, podemos entender melhor a análise interpretando a matriz da imagem como um vetor de informações, onde cada linha é um componente e cada coluna diferentes atributos desse componente.

Com a imagem inicial $X$, na eq. (2), removemos de cada componente a média dos atributos (representada na matriz $\overline{X_j}$), de forma a obtermos uma média zero individualmente. Com isso, podemos obter a matriz de covariância $X_n$, como em eq. (2b).

$$X_n = X - \overline{X_j} \tag{2a}$$

$$\Sigma = X_n X_n^T \tag{2b}$$

Com a matriz de covariância da eq. (2b), encontramos e ordenamos os autovetores decrescentemente pelos seus autovalores na eq. (3). Disso, isolamos os $k$ primeiros componentes dos autovetores.

$$\lambda, V = \texttt{sort}(\texttt{eig}(\Sigma)) \tag{3a}$$

$$V_k = V_{[:,0:k]} \tag{3b}$$

Utilizando os $k$ primeiros componentes, refazemos a matriz original na eq. (4) utilizando os $k$ primeiros autovetores e a matriz sem as médias, somando as médias removidas e finalmente produzindo a imagem final $X_f$.

$$\hat{X} = V_k^T X_n \tag{4a}$$

$$X_f = (V_k \hat{X}) + \overline{X_j} \tag{4b}$$

### 2.3 Avaliando resultados

Agora, com as duas formas de se produzir uma imagem utilizando menos componentes, podemos avaliar a compressão. Para isso, salvamos cada imagem utilizando *OpenCV*, calculamos a taxa de compressão e RMSE (*Root mean square error*). Esses dados são impressos em um arquivo e podem ser comparados apenas dentro de uma mesma imagem (devida natureza de RMSE).

Os testes foram automatizados através de um *script* `bash`. Fazemos para cada imagem alguns valores de $k$: 32, 64, 128, 192 e 256, produzindo uma imagem de saída e um relatório diferente.

## 3 Resultados

Como explicitado em aula, podemos comprovar a dificuldade para escolher um valor $k$. Há imagens em que visualmente não podemos notar artefatos com $k$ pequeno e outras que fazem muita diferença, mesmo em dimensões parecidas. Podemos perceber isso na comparação das imagens `monalisa` e `watch-menor` na fig. 1, cujas dimensões são parecidas e número de *pixels* diferem em cerca de apenas 2 mil.

(a) `monalisa` original

(b) `monalisa` $k = 64$

(c) `watch-menor` original

(d) `watch-menor` $k = 64$

Figura 1: Comparativo entre duas imagens de dimensões próximas

## 4 Conclusão