Projeto
MAC460 - Aprendizagem Computacional

Fernando Omar Aluani 6797226
Renan Teruo Carneiro 6514157

1 O Problema

O problema de *Landmark Detection* é um problema de classificação, onde a entrada é uma imagem e a saída é um rótulo (um valor numérico ou uma *string*) que representa o nome do monumento retratado na imagem.

Um dos grandes desafios desse problema é a representação dos dados da imagem. O formato padrão de representação pelos pixels da imagem não ajuda muito, pois eles não dizem muito sobre o conteúdo dela. É necessário extrair dados que sejam mais úteis.

2 Métodos Utilizados

A solução do problema consiste em treinar um classificador com um conjunto de imagens, sabendo suas respectivas classes, para então usar esse classificador para prever a classe de uma dada imagem.

Para conseguir isso, usamos o modelo *Bag Of Words*, que define um vocabulário de palavras de tal forma que o vetor de características de uma imagem é um histograma que marca a ocorrência de cada palavra em um objeto. A definição do vocabulário para imagens é dada pela detecção de pontos especiais nas imagens, chamados *KeyPoints*, e seus respectivos vetores de descrição.

Com a biblioteca **OpenCV**, implementar essa solução consiste em usar algumas classes que se relacionam entre si:

- ***FeatureDescriptor*:** calcula os *KeyPoints* para uma dada imagem;
- ***DescriptorExtractor*:** calcula os descritores dos *KeyPoints*;
- ***BOWKMeansTrainer*:** usa *KMeans* para calcular o vocabulário a partir dos descritores das imagens;
- ***BOWImgDescriptorExtractor*:** sabendo o vocabulário, calcula o histograma de uma imagem;
- ***Classifier*:** realiza o treinamento e teste em si. Para treinar ele usa os histogramas das imagens e o rótulo de suas respectivas classes. E para o teste ele devolve o rótulo de um dado histograma;

3 Resultados

Existem várias possibilidades de *FeatureDescriptor*, *DescriptorExtractor* e *Classifier* que podemos usar, e após alguns testes para descobrir quais funcionavam com o nosso programa optamos pelos seguintes:

- ***FeatureDescriptor*:** *FAST, STAR e SIFT*;
- ***DescriptorExtractor*:** *SIFT e SURF*;
- ***Classifier*:** *Bayes, kNN e Árvore de Decisão*

Aqui estão os resultados com cada combinação de *Classifier*, *FeatureDescriptor* e *DescriptorExtractor* (notando que os nomes estão nessa mesma ordem):

3.1 Bayes - FAST - SIFT

	Classe Prevista					
	panteão	muralha-da-china	cristo	torre-eiffel	taj-mahal	piramides
panteão	8	0	0	0	0	0
muralha-da-china	0	9	1	0	0	0
cristo	0	3	8	0	0	0
torre-eiffel	1	0	1	8	0	0
taj-mahal	0	0	0	0	9	0
piramides	0	0	2	0	0	8

3.2 Bayes - STAR - SURF

	Classe Prevista					
	panteão	muralha-da-china	cristo	torre-eiffel	taj-mahal	piramides
panteão	8	0	0	0	0	0
muralha-da-china	3	3	3	1	0	0
cristo	0	1	9	1	0	0
torre-eiffel	3	1	3	3	0	0
taj-mahal	0	1	1	1	6	0
piramides	2	0	2	1	3	2

3.3 Bayes - FAST - SURF

	Classe Prevista					
	panteão	muralha-da-china	cristo	torre-eiffel	taj-mahal	piramides
panteão	6	0	0	0	2	0
muralha-da-china	0	6	3	1	0	0
cristo	0	0	11	0	0	0
torre-eiffel	1	1	7	1	0	0
taj-mahal	0	0	0	0	9	0
piramides	0	0	1	0	0	9

3.4 Bayes - SURF - SIFT

	Classe Prevista					
	panteão	muralha-da-china	cristo	torre-eiffel	taj-mahal	piramides
panteão	8	0	0	0	0	0
muralha-da-china	0	7	1	0	0	2
cristo	0	4	7	0	0	0
torre-eiffel	0	0	0	10	0	0
taj-mahal	0	0	0	0	9	0
piramides	0	2	2	2	1	3

3.5 Bayes - STAR - SIFT

	Classe Prevista					
	panteão	muralha-da-china	cristo	torre-eiffel	taj-mahal	piramides
panteão	7	0	1	0	0	0
muralha-da-china	0	7	2	1	0	0
cristo	0	1	10	0	0	0
torre-eiffel	1	0	0	8	1	0
taj-mahal	0	0	0	0	9	0
piramides	0	1	1	0	1	7

3.6 Bayes - SURF - SURF

	Classe Prevista					
	panteão	muralha-da-china	cristo	torre-eiffel	taj-mahal	piramides
panteão	7	0	1	0	0	0
muralha-da-china	0	7	3	0	0	0
cristo	0	3	8	0	0	0
torre-eiffel	1	0	1	8	0	0
taj-mahal	0	0	1	1	7	0
piramides	0	1	0	0	3	6

3.7 kNN - FAST - SIFT

	Classe Prevista					
	torre-eiffel	panteão	muralha-da-china	cristo	taj-mahal	piramides
torre-eiffel	9	0	1	0	0	0
panteão	0	8	0	0	0	0
muralha-da-china	0	1	8	1	0	0
cristo	0	0	1	10	0	0
taj-mahal	1	0	0	0	8	0
piramides	1	1	4	0	0	4

3.8 kNN - STAR - SURF

	Classe Prevista					
	torre-eiffel	panteão	muralha-da-china	cristo	taj-mahal	piramides
torre-eiffel	2	5	1	1	1	0
panteão	0	8	0	0	0	0
muralha-da-china	2	7	1	0	0	0
cristo	4	2	1	4	0	0
taj-mahal	0	6	0	0	3	0
piramides	2	3	1	0	3	1

3.9 kNN - STAR - SIFT

	Classe Prevista					
	torre-eiffel	panteão	muralha-da-china	cristo	taj-mahal	piramides
torre-eiffel	8	0	0	0	2	0
panteão	0	8	0	0	0	0
muralha-da-china	4	2	4	0	0	0
cristo	1	0	0	10	0	0
taj-mahal	0	1	0	0	8	0
piramides	1	0	2	0	1	6

3.10 kNN - SURF - SURF

	Classe Prevista					
	torre-eiffel	panteão	muralha-da-china	cristo	taj-mahal	piramides
torre-eiffel	6	1	0	1	2	0
panteão	0	8	0	0	0	0
muralha-da-china	0	2	6	2	0	0
cristo	0	0	5	5	1	0
taj-mahal	0	3	0	0	6	0
piramides	1	2	1	1	0	5

3.11 kNN - FAST - SURF

	Classe Prevista					
	torre-eiffel	panteão	muralha-da-china	cristo	taj-mahal	piramides
torre-eiffel	5	2	0	2	1	0
panteão	0	7	0	0	1	0
muralha-da-china	3	1	5	0	1	0
cristo	2	1	0	8	0	0
taj-mahal	1	2	0	1	5	0
piramides	0	1	0	1	0	8

3.12 kNN - SURF - SIFT

	Classe Prevista					
	torre-eiffel	panteão	muralha-da-china	cristo	taj-mahal	piramides
torre-eiffel	10	0	0	0	0	0
panteão	0	8	0	0	0	0
muralha-da-china	0	0	8	1	1	0
cristo	0	0	8	3	0	0
taj-mahal	0	0	0	0	9	0
piramides	4	0	3	1	0	2

3.13 DecisionTree - STAR - SIFT

	Classe Prevista					
	torre-eiffel	panteão	muralha-da-china	cristo	taj-mahal	piramides
torre-eiffel	9	0	0	1	0	0
panteão	0	7	0	0	1	0
muralha-da-china	0	1	9	0	0	0
cristo	1	0	7	2	0	1
taj-mahal	0	5	0	0	3	1
piramides	2	0	4	0	0	4

3.14 DecisionTree - SURF - SURF

	Classe Prevista					
	torre-eiffel	panteão	muralha-da-china	cristo	taj-mahal	piramides
torre-eiffel	9	1	0	0	0	0
panteão	1	6	0	0	1	0
muralha-da-china	1	0	6	1	0	2
cristo	4	0	4	3	0	0
taj-mahal	1	2	0	2	3	1
piramides	5	0	0	0	0	5

3.15 DecisionTree - FAST - SURF

	Classe Prevista					
	torre-eiffel	panteão	muralha-da-china	cristo	taj-mahal	piramides
torre-eiffel	8	1	1	0	0	0
panteão	1	3	0	0	4	0
muralha-da-china	1	3	6	0	0	0
cristo	5	1	4	0	0	1
taj-mahal	2	4	0	0	3	0
piramides	1	1	0	0	0	8

3.16 DecisionTree - SURF - SIFT

	Classe Prevista					
	torre-eiffel	panteão	muralha-da-china	cristo	taj-mahal	piramides
torre-eiffel	4	0	2	3	0	1
panteão	0	7	0	0	1	0
muralha-da-china	0	0	6	2	1	1
cristo	0	1	8	2	0	0
taj-mahal	1	1	0	0	7	0
piramides	5	0	0	0	0	5

3.17 DecisionTree - FAST - SIFT

	Classe Prevista					
	torre-eiffel	panteão	muralha-da-china	cristo	taj-mahal	piramides
torre-eiffel	1	2	0	5	0	2
panteão	0	7	0	0	1	0
muralha-da-china	2	0	4	3	0	1
cristo	0	1	2	6	0	2
taj-mahal	1	1	0	1	6	0
piramides	0	0	2	0	0	8

3.18 DecisionTree - STAR - SURF

	Classe Prevista					
	torre-eiffel	panteão	muralha-da-china	cristo	taj-mahal	piramides
torre-eiffel	0	0	8	0	0	2
panteão	0	0	7	0	0	1
muralha-da-china	0	0	9	0	0	1
cristo	0	0	10	0	0	1
taj-mahal	0	0	9	0	0	0
piramides	0	0	3	0	0	7

4 Processo de Desenvolvimento

Primeiramente tentamos implementar em C++ um unico classificador (uma única combinação de *FeatureDescriptor*, *DescriptorExtractor* e *Classifier*) e testá-lo, para ver se nosso algoritmo estava funcionando. Inicialmente seguimos um tutorial para implementar o classificador, alterando-o de acordo com nossas necessidades.

Tivemos alguns problemas com os formatos das matrizes do OpenCV, alguns problemas ao escrever o algoritmo básico (traduzindo do tutorial) e a dificuldade em testar o algoritmo pois o processo é lento.

Eventualmente conseguimos fazer esse classificador funcionar. Então nós generalizamos o código do classificador para que ele rodasse com combinações diferentes de *FeatureDescriptor*, *DescriptorExtractor* e *Classifier*, escolhidas dinâmicamente em *run-time*. Enquanto o próprio OpenCV tem métodos para criação dinâmica de *FeatureDescriptor* e *DescriptorExtractor* passando somente uma *string* com o nome, para generalizar o *Classifier* tivemos que usar *templates* de C++, e um pouco de código para saber qual template usar de acordo com o valor passado na linha de comando.

A partir daí o maior problema foi os longos períodos de tempo que levavam para executar o treinamento e teste de um classificador. Testamos várias combinações. Descobrimos que algumas não funcionavam (usando o *DescriptorExtractor ORB*, por exemplo), e não conseguimos fazer alguns *Classifier*s funcionarem (como o *CvRTrees*, por exemplo).

Depois dos testes, decidimos usar somente aqueles mencionados anteriormente (em Resultados). Fizemos 2 scripts (em Python) para automatizar a execução e registro dos resultados das várias possibilidades de classificadores que tínhamos, e usamos eles para colher os resultados apresentados neste relatório.

5 Executáveis

Aqui vai uma breve descrição dos executáveis que desenvolvemos no projeto:

5.1 LandmarkDetector

Este é o programa principal do projeto, é a implementação de um classificador generalizado em C++. Por usar internamente de coisas específicas do Linux, o programa só funcionará nessa plataforma.

Para compilar, execute o *CMake* para gerar um *Makefile*, e então use o *make* para compilar o programa. O CMake procura o OpenCV instalado na máquina.

Para executar, rode:

```
./LandmarkDetector <CLASSIFIER> <DESCRIPTOR> <EXTRACTOR> <TRAIN> <TEST>
```

Onde:

- CLASSIFIER: um valor que dita qual *Classifier* usar. Pode ser *BAYES*, *KNN* ou *DTREE*(árvore de decisão).
- DESCRIPTOR: especifica qual *FeatureDescriptor* usar. As opções dependem do OpenCV, refira-se a doc do OpenCV.
- EXTARCTOR: especifica qual *DescriptorExtractor* usar. As opções dependem do OpenCV, refira-se a doc do OpenCV.
- TRAIN: caminho para o diretório do conjunto de treinamento. Este diretório deve conter outras subpastas, onde o nome da subpasta é o rótulo de uma classe, e as imagens dentro dessa subpasta são as imagens para treinar o classificador para tal classe.
- TEST: caminho para o diretório do conjunto de testes. Este diretório deve conter outras subpastas, onde o nome da subpasta é o rótulo de uma classe, e as imagens dentro dessa subpasta são as imagens para testar o classificador.

O programa imprime na saída padrão o andamento do treinamento, e depois imprime o resultado do teste de cada imagem no conjunto de testes.

5.2 classifierTest.py

Pequeno script Python que roda o LandmarkDetector para todas possibilidades de classificadores que escolhemos, e redireciona o output do classificador para arquivos de log que são posicionados numa pasta *logs*, localizada na mesma pasta onde o script e o LandmarkDetector estão.

O script só roda o classificador e salva o log para as possibilidades que não tem um log já criado.

Para executá-lo:

```
./classifierTest.py [CLASSIFIER]
```

Onde *CLASSIFIER* é o valor para ser passado para o campo de mesmo nome ao executar o LandmarkDetector. Se esse argumento opcional for passado, o script só rodara as combinações para um único *Classifier* (o que foi especificado no argumento). Se esse argumento não for passado, ele irá executar todas possibilidades possíveis.

5.3 logParser.py

Pequeno script Python que lê os logs criados pelo classifierTest.py, e gera relatórios sobre a eficiência dos classificadores. Ele gera o relatório para cada classificador na própria pasta *logs*, com o nome do log original mais o sufixo *_result*. Ele também imprime na saída padrão qual combinação foi a que teve o melhor resultado.

O script também gera *snippets* de código LaTeX(numa pasta *doc*) para criação da tabela da matriz de confusão de cada classificador. Nós usamos tais *snippets* para gerar as tabelas dos resultados apresentados aqui.

6 Referências

As principais referências que usamos foram:

Documentação do OpenCV

Tutorial sobre classificação de objetos usando Bag of Words e OpenCV

Mas também consultamos o fórum da disciplina no PACA e fizemos algumas buscas no *Google* quando tivemos problemas com algumas partes específicas do OpenCV.