Universidade Federal do Rio Grande do Norte
Instituto Metrópole Digital

IMD0040 - Linguagem de Programação II

Especificação do Projeto Final
2019.2

Proposta: Detecção de Pessoas em Imagens através do uso de Inteligência Computacional

1 Introdução

A análise de imagens faz parte do cotidiano da maioria de nós. Observamos paisagens, fotos da família, notícias na televisão, etc, e com base nas imagens apresentadas em cada contexto formamos o nosso raciocínio sobre a respectiva questão. Com base na mesma lógica, várias áreas da computação tentam se apropriar de informações contidas em imagens para extrair conhecimento e prover informações para tomada de decisão. Esse fato acontece em diversas tarefas computacionais conhecidas, como, identificação facial, identificação de movimentos, identificação de objetos, dentre outros.

Este projeto lida com este escopo de atuação, ou seja, utiliza imagens como base para tomada de decisão em sistemas. Para tal, propõe-se a criação de um sistema que terá como entrada uma imagem, e como saída, a informação da existência ou não de pessoas contidas na mesma. A partir daí, utilizar essa informação de forma direcionada para o escopo de projeto escolhido pelo grupo (e.g. ligar/desligar aparelho de ar condicionado, ativa/desativar semáforos, etc.).

2 Objetivo e Metodologia

2.1 Objetivo Geral

Implementar um sistema que identifique se há pessoa(s) em uma imagem e utilize essa informação para tomada de decisão. Por exemplo, um sistema de controle de aparelhos de ar-condicionado, que liga/desliga os aparelhos de acordo com a presença de pessoas no ambiente. O sistema terá duas partes essenciais:

- a **primeira**, capaz de receber uma imagem como entrada, tratá-la e transformá-la em um vetor de características, além de buscar, dentre as imagens contidas no dataset disponibilizado as "k" imagens mais próximas (semelhantes) à respectiva imagem (de entrada). Além disso, retornar, com base nas características das "k" imagens mais próximas, uma resposta (rótulo) que representará a presença ou não de pessoas na referida imagem.
- a **segunda**, o sistema a ser desenvolvido utiliza uma interface gráfica (em JavaFx) para receber imagens de entrada que serão tratadas e transformadas (primeira parte) e para informar se a imagem fornecida tem ou não pessoas. A partir dessa informação de saída, o sistema deverá fazer algum tipo de controle baseado na mesma.

2.2 Metodologia e Descrição Geral do Sistema

O sistema a ser desenvolvido deverá contemplar os seguintes itens e respectivas funcionalidades:

1. Implementar os Módulos de leitura do dataset (Dataset) e análise dos dados (Core):
   (a) Ler os dados extraídos de várias imagens contendo ou não pessoas (dataset disponibilizado);

(b) Transformar o dataset em uma coleção de objetos, sendo cada objeto a representação de uma imagem;
(c) Implementar o algoritmo ***K-NN*** *(K-NN)* para calcular quais os "k" objetos mais próximos a imagem de entrada. O valor de "k" poderá variar entre 1 e 10 e será fornecido pelo próprio sistema.

2. Implementar o módulo de extração de características de imagens de acordo com o *Tutorial OpenCV* e observar os seguintes requisitos:
   (a) Reduzir o tamanho da sua imagem para **64 X 128 pixels**;
   (b) Transformar cada imagem de entrada em um objeto com as **1000 primeiras** características disponíveis para o ***Histogram of Object Gradients*** **(HOG)**;
   (c) Utilizar **OpenCV** *(Tutorial OpenCV)*.
   (d) Utilizar os mesmos parâmetros apresentados no tutorial para extração do *HOG*;
3. Implementar um módulo que utilizará os dois módulos anteriores para tomada de decisão.
   (a) Implementar alguma forma de entrada para uma nova imagem, como câmera, pasta de arquivos, streaming, vídeo, etc.. Sendo a entrada um vídeo, câmera ou *streaming*, capture a imagem do momento para realizar a análise, salvando-a em formato adequado (**.png**).
   (b) Utilizar o módulo de extração de características para transformar a imagem obtida pelo sistema;
   (c) Utilizar o módulo "Core" para buscar quais os "k" objetos mais próximos (K-NN) à nova imagem e definir se há ou não a presença de pessoas de acordo com as imagens (objetos) mais próximas.
   (d) Utilizar essa informação como entrada do módulo de controle e simular ou demonstrar a tomada de decisão do sistema através da Interface gráfica.

3 Conceitos Relacionados

3.1 *Dataset*

Um *dataset* é uma coleção de dados organizados, geralmente representados em um arquivo. Um de seus formatos mais comuns é o *.csv*. A Figura 1 mostra a estrutura básica dos dados contidos num dataset além de alguns dos principais termos utilizados no contexto de classificação de dados (tarefa realizada, por exemplo, pelo algoritmo K-NN). Uma imagem (**Instância**) corresponderá a uma linha do arquivo csv; as colunas corresponderão às características (**Atributos**) das imagens e, por fim, a última coluna é um atributo especial que indicará a categoria (**Rótulo**) da imagem. Para o dataset disponibilizado, este rótulo foi adicionado manualmente observando a imagem (Sua tarefa envolverá dar esse rótulo de forma automatizada para novas imagens).

Figura 1: Exemplo de um arquivo csv para a identificação de pessoas numa imagem.

O *dataset* que será utilizando na aplicação possui 1000 atributos com os valores **numéricos** e um rótulo que pode assumir 2 estados (*Person* ou *notPerson*) identificando se a imagem possui pessoa(s) ou não.Este dataset é composto por um subconjunto das imagens contidas no dataset original publicado no trabalho David Geronimo Gomez. *A global approach to vision-based pedestrian detection for advanced driver assistance systems*. Universitat Autònoma de Barcelona, 2010.

A partir das informações contidas no dataset disponibilizado, será aplicado o algoritmo do K-NNpara realizar a verificação se existe pessoas ou não em novas instâncias (novas imagens).

3.2 Algortimo K-NN

O k vizinhos próximos (K-NN, do inglês *K-Nearest Neighbor*) faz uso de funções de distância para ranquear as instâncias mais próximas da instância que está sendo passada por parâmetro. Considerando a distribuição dos dados apresentada na Figura 2 onde, **P** representa uma imagem com uma pessoa e **NP** representa uma imagem que não possui uma pessoa. Num dado momento chega uma nova imagem para realizar a identificação (Figura 3) representada por um △.

A Figura 4 demonstra o processo de calcular a distância de todas as imagens para a nova imagem e selecionar uma quantidade k de imagens – neste caso 1 imagem – para identificar o rótulo desta (apresentado na Figura 5).

Figura 2: Distribuição dos dados.

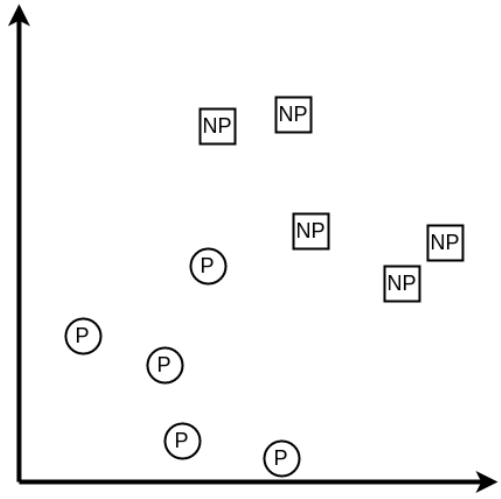


Figura 3: Nova imagem para identificar o rótulo.

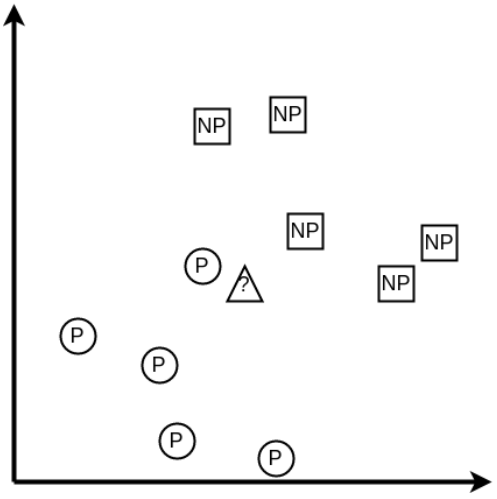


Figura 4: Calculando a distância de todas as imagens conhecidas para a nova imagem.

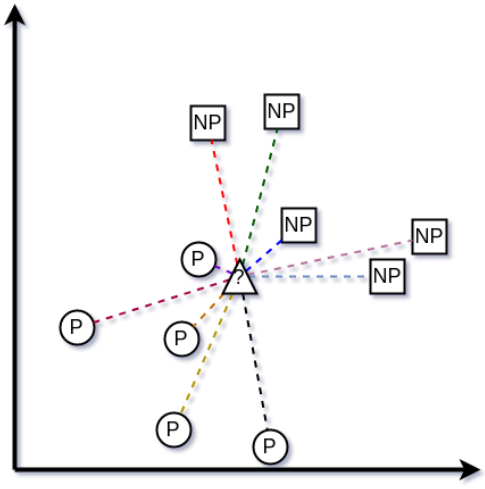
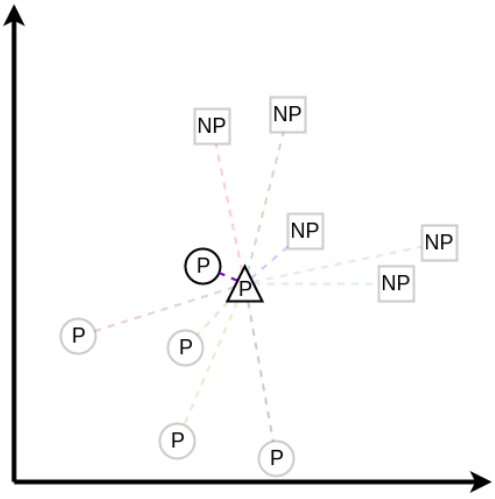


Figura 5: Selecionando a imagem mais próxima e aplicando o rótulo dela na nova imagem.



O Algoritmo 1 mostra, de forma resumida, o fluxo básico do algoritmo do K-NN. Na Linha 2 é calculada a diferença por meio das Equações 4-6. Após calcular essa diferença para cada uma das imagens, é necessário ranquear as imagens na ordem crescente, selecionando k imagens. Por fim, o rótulo mais frequente das k imagens selecionas é utilizado na imagem da presente iteração.

Pseudocódigo 1: K-NN

entrada $Imagens = i_1, \ldots, i_n$

1: **para cada** imagem em $Imagens$ **faça**
2: Calcular a diferença da presente imagem para as demais;
3: Ranquear as imagens em ordem de relevância (de acordo com a métrica);
4: Selecionar as k imagens mais próximas;
5: Observar o rótulo mais frequente;
6: Rotular a imagem atual utilizando rótulo mais frequente.
7: **fim para**

Existem diversas métricas utilizadas na realização do cálculo da distância para o algoritmo do K-NN, dentre elas: a distância Euclidiana, a distância de *Chebychev* também conhecida como valor máximo, a distância de *Manhattan* que é calculada através da diferença entre os pares de pontos dados, e outras. Avaliando um espaço de 1000 dimensões (o arquivo que é gerado após o processamento das imagens); sejam $x = \{x_1, x_2, \ldots, x_{1000}\}$ e $z = \{z_1, z_2, \ldots, z_{1000}\}$ duas imagens distintas, a distância entre esses dois objetos utilizando a distância euclidiana é apresentada na Equação 4. Por sua vez, a Equação 5 apresenta a distância de *Manhattan* e, por fim, a Equação 6 denota a distância de *Chebyshev*.

$$\sqrt{\sum_{i=1}^{1000}(x_i - z_i)^2} \tag{1}$$

$$\sum_{i=1}^{1000}|x_i - z_i| \tag{2}$$

$$\mathbf{max}((z_1 - x_1), (z_2 - x_2), \cdots, (z_{1000} - x_{1000})) \tag{3}$$

3.3 Medidas de Distância

Existe na literatura vários algoritmos e funções capazes de calcular a distância entre dois pontos. Contudo, será apresentado um grupo com apenas 3 distâncias, sendo elas: distância Euclidiana, Manhattan e Chebychev.

3.3.1 Distância Euclidiana

A distância Euclidiana é dada pela menor distância entre dois pontos, que pode ser provada utilizando a teorema de Pitágoras repetidamente.

Sejam $x = \{x_1, x_2, \ldots, x_n\}$ e $z = \{z_1, z_2, \ldots, z_n\}$ duas imagens distintas, onde n é a quantidade de características do objeto. A distância entre esses dois objetos utilizando a distância Euclidiana é dada pela equação:

$$\sqrt{\sum_{i=1}^{n}(x_i - z_i)^2} \tag{4}$$

3.3.2 Distância de Manhattan

A distância Manhattan é uma métrica dada pela menor distância entre dois pontos de um sistema cartesiano com a soma das diferenças absolutas entre as suas coordenadas. Por exemplo, imagine que um carro irá

fazer um trajeto de um ponto A até um ponto B de uma cidade, como na imagem abaixo, a menor distância que o carro pode percorrer é descrita pelas rotas vermelha, azul e amarela.

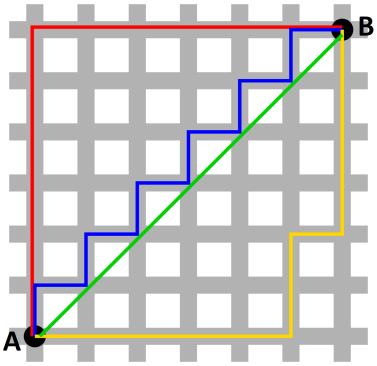


Figura 6: Exemplo da distância de Manhattan

Portanto no escopo do nosso projeto, considere $x = \{x_1, x_2, \ldots, x_n\}$ e $z = \{z_1, z_2, \ldots, z_n\}$ duas imagens distintas, onde n é a quantidade de características do objeto. O cálculo da distância de Manhattan entre esses dois objetos é dada pela equação:

$$\sum_{i=1}^{n} |x_i - z_i| \tag{5}$$

3.3.3 Distância de Chebychev

A distância de Chebychev, também chamada de métrica máxima, é uma métrica calculada em um espaço de vetores, onde a distância entre dois vetores é a maior de suas diferenças entre suas dimensões de coordenadas.

Sejam $x = \{x_1, x_2, \ldots, x_n\}$ e $z = \{z_1, z_2, \ldots, z_n\}$ duas imagens distintas, onde n é a quantidade de características do objeto. O cálculo da distância de Chebychev entre esses dois objetos é dada pela equação:

$$\mathbf{max}((z_1 - x_1), (z_2 - x_2), \cdots, (z_n - x_n)) \tag{6}$$

4 Requisitos pertinentes à disciplina

Apesar de ser um trabalho que permite diversas abordagens para a resolução do problema supracitado, é necessário observar que, para fins de avaliação da disciplina em curso, serão considerados todos os conceitos vistos durante o semestre como base dessa avaliação. Para tal, será obrigatório, portanto, o uso dos seguintes conceitos:

- Paradigma Orientado à Objetos;
  - Objetos e classes, atributos e métodos;
- Encapsulamento;
  - Utilização de padronização de acesso e modificação através dos métodos getters e setters;
- Modularização e padronização;
  - Nomeação de classes e métodos segundo padrões;
  - Separação das classes em pacotes;
  - Design de Classes (responsabilidade e coesão);
- Herança;
- Polimorfismo;
- Classes abstratas;

- Interfaces;
- Padrões de projeto;
  - Utilização de alguns padrões de projeto de forma justificada;
- Tratamento de exceções;
- Interface Gráfica;
  - Utilização de JavaFx para interface Gráfica;

5 Entrega e Avaliação

Seu grupo deverá submeter, via SIGAA, um arquivo compactado contendo:

- Código fonte do sistema desenvolvido, incluindo um arquivo README.txt contendo as instruções de como compilar o projeto.
- Relatório técnico, contendo pelo menos as seguintes seções:
  - **Introdução** contendo uma breve descrição do problema abordado e do que será apresentado no relatório;
  - **Descrição da abordagem** utilizada para solução do problema, descrevendo o projeto OO (diagrama de classes) e destacando as decisões de projetos tomadas, bem como identificando padrões de projetos aplicados;
  - **Descrição geral** das estruturas de dados utilizadas, com explicação dos algoritmos utilizados;
  - **Conclusão**;
  - **Referências**;
- Arquivo em PDF contendo a apresentação do projeto.

O relatório deverá ser feito seguindo o template da Sociedade Brasileira de Computação (SBC), que pode ser encontrado na seção de Material de Apoio. A avaliação do projeto será dada através da tabela de pontuação (Tabela 1):

Codificação	Modularização e padronização	0,5
	Herança e Polimorfismo	0,5
	Classes Abstratas ou Interfaces	0,5
	Tratamento de Exceções	0,5
	Interface Gráfica	1,0
Apresentação	Relatório	2,0
	Apresentação	1,0
Objetivos	Pré-processamento das imagens	2,0
	Implementação do k-NN	1,0
	Implementação de uma métrica de distância	1,0
	Plus (ponto extra)	
Total de pontos		10,0

Tabela 1: Tabela de pontuação para correção

5.1 Apresentação

Cada grupo terá 20 minutos para realizar a apresentação. Nesta apresentação, os grupos deverão demonstrar o funcionamento do programa desenvolvido. Além disso, deverão estar aptos a responder questões sobre o desenvolvimento do projeto.

Importante: não será dada uma única nota ao grupo! Cada componente do grupo receberá uma nota de acordo com seu desempenho durante a apresentação.

O programa será avaliado como um todo, ou seja, os requisitos não receberão pontuação individualmente. Dessa forma, a falta de um ou mais requisitos acarretará na perda de pontos, que poderá ser compensada (não totalmente, claro) através de outros componentes bem desenvolvidos. Componentes adicionais serão muito bem vistos, desde que implementados de maneira racional.

6 Material de apoio

- **O que é K-NN?**
  http://bit.ly/31X8RHp
- **O que é OpenCV?**
  https://opencv.org/
- **Dataset**
  (http://bit.ly/2IsDKMg)
- **JavaFx**
  https://openjfx.io/
- **Template SBC**
  http://bit.ly/2MiQCFU

Referências

Gomez, David Geronimo. *A global approach to vision-based pedestrian detection for advanced driver assistance systems*. Universitat Autònoma de Barcelona, 2010.

Anexos

7 Tutorial OpenCV

OpenCV é uma biblioteca *opensource* de visão computacional. Ela implementa *builds* para diversas linguagens de programação, C++, Python e Java. Com ela o usuário pode realizar de maneira simples as clássicas operações em imagens.

7.1 Instalação

7.1.1 Instalação Windows

A instalação do OpenCV no Windows é muito simples. Tudo que é necessário é fazer o download do instalador no site do OpenCV.

7.1.2 Instalação Linux

Para instalar o OpenCV no Linux é necessário fazer a instalação de suas dependências primeiro. Para tal, basta rodar os seguintes comandos:

```
(compiler) sudo apt-get install build-essential
(required) sudo apt-get install cmake git libgtk2.0-dev
(required) sudo apt-get install pkg-config libavcodec-dev
(required) sudo apt-get install ibavformat-dev libswscale-dev
```

Agora, com o todas as dependências e o *builder* CMake instalado você precisa baixar o código fonte do OpenCV e compilar. Isso pode ser feito de duas maneiras: clonando o repositório no GitHub, ou baixando a ultima versão estável neste site. Para clonar o repositório do GitHub e entra na paste do código fonte rode:

```
cd ~/<Meu diretório de trabalho>
git clone https://github.com/opencv/opencv.git
cd opencv
```

No diretório do código fonte vamos criar uma pasta chamada `build` para compilar o projeto

```
~/opencv $ mkdir build;cd build
```

Na pasta build vamos utilizar a ferramenta CMake para gerar um *Makefile* que instalará as versões do OpenCV que queremos.

Para gerar um *Makefile* que instala a versão Java vamos utilizar o comando :

```
/build $ cmake -D CMAKE_BUILD_TYPE=Release -D CMAKE_INSTALL_PREFIX=/usr/local BUILD_SHARED_LIBS=OFF ..
```

Atenção! No *output* do último comando devem estar presentes as seguintes linhas :

```
--   Java:
--     ant:                 /usr/bin/ant (ver 1.10.5)
--     JNI:                 /usr/lib/jvm/java-8-openjdk-amd64/include
                            /usr/lib/jvm/java-8-openjdk-amd64/include/linux
                            /usr/lib/jvm/java-8-openjdk-amd64/include

--     Java wrappers:           YES
--     Java tests:              YES
--
```

Nessa linha o CMake mostra o caminho para o Java ant, que é uma ferramenta utilizada para construir projetos Java, similar ao Maven, e para o JNI, que é o caminho para o JDK da sua máquina, no caso de faltar algum do dois a instalação do OpenCV não será concluída. Para resolver esse problema podemos simplesmente instalá-los com o comando :

```
sudo apt-get install ant openjdk-8-jdk
```

Agora podemos novamente rodar:

```
/build $ cmake -D CMAKE_BUILD_TYPE=Release -D CMAKE_INSTALL_PREFIX=/usr/local BUILD_SHARED_LIBS=OFF ..
```

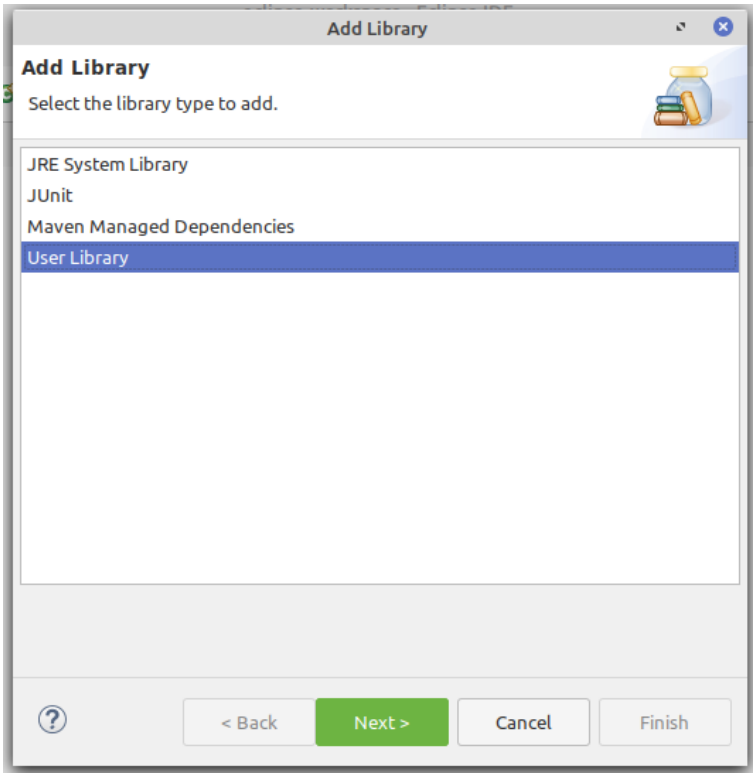
E o *output* deve estar similar ao do exemplo.
Agora basta rodar:

```
/build $ make -j4
```

7.2 Utilizando o OpenCV no Ecplise

Como o processo é bem similar tanto no Linux quanto no Windows, mudando apenas a extensão de um arquivo, vamos seguir com os dois processos ao mesmo tempo.

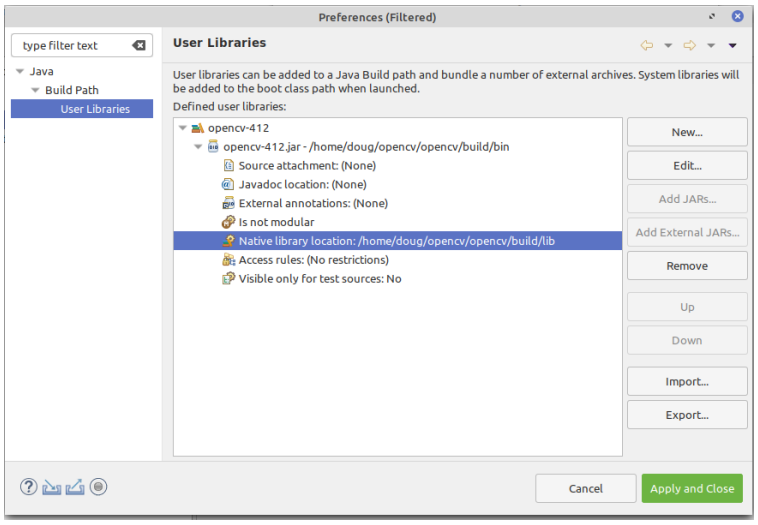
Para começar crie um projeto Java comum. Com projeto criado, clique com o botão direito do mouse e vá para a opção `build path -> add libraries` em seguida clique em `user library` como na imagem abaixo:



Em seguida clicando em `user libraries` novamente. Agora na tela de bibliotecas do usuário clicando em `New` e damos o nome Opencv-4.12

Com a nossa biblioteca selecionada clicamos em `Add External Jar` e selecionamos o caminho para o caminho do aquivo Jar compilado do Opencv, no Linux esse arquivo fica na pasta `.../opencv/build/bin`, já no Windows esse arquivo fica em `C:opencv\build\java`.

Finalmente precisamos adicionar o arquivo que de *link* da biblioteca, no Windows é a um arquivo **.dll** e no Linux é um arquivo **.so**. No Linux esse arquivo fica em `.../opencv/build/lib`, já no Windows ele fica em `C:opencv\build\java\x64`. Para isso é só editar o campo *highlighted* na imagem a seguir:



Finalmente, para adicionar a sua biblioteca ao seu projeto, clique com o botão direito no seu projeto vá em `Build Path -> Add Libraries` depois clique em `user libraries` e selecione a sua biblioteca. Agora seu projeto está pronto para utilizar toda funcionalidade do OpenCV.

7.3 Comandos básicos

Bom, para começar, na classe principal do seu programa deve estar presente a seguinte linha:

```
import org.opencv.core.Core;
public class Exemplo {
        static {
            System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
         }
}
```

Para ler uma imagem em *grayscale* utilize o seguinte código

Um ponto importante sobre tratamento de imagens, quanto mais canais (informação sobre cor e transparência) mais pesada a imagem e mais demoradas são as operações, para resolver esse problema podemos lidar com imagens com um único canal, essa imagens são as imagens *gray scale* para importar uma imagem em formato *gray scale* e guardá-la em um objeto *Mat* (objeto responsável por representar as imagens no OpenCV) são necessárias as seguintes instruções descritas nas próxima subseções.

7.4 HOG no OpenCV

.

Para extrair características sobre as formas presentes na imagem você deve utilizar o objeto HOGDescriptor presente no OpenCV. Nesse objeto etá presente o método *compute* que gera uma lista de *float* que descrevem a imagem.

O primeiro passo para realizar as extração de características é redimensionar a imagem de entradas:

```
import org.opencv.imgcodecs.Imgcodecs;
import org.opencv.imgproc.Imgproc;
import org.opencv.objdetect.HOGDescriptor;
public class HogExtractor {
        public static void main(String[] args) {
                HOGDescriptor hog = new HOGDescriptor();
                Mat img = new Mat();
                MatOfFloat features = new MatOfFloat();
```

```
                img = Imgcodecs.imread("caminho/para/imagem", Imgcodecs.IMREAD_GRAYSCALE);
                Imgproc.resize(img, img, new Size(64,128), 0.5, 0.5, Imgproc.INTER_LINEAR);
                hog.compute(img,features);
                List<Float> arrayOfFeatures = features.toList();
        }

}
```

O método `toList()` do MatOfFloats permite transformar que o programador trate os atributos como uma *List* do *Java Collections* facilitando o cálculo por exemplo da distancia entre as características de duas imagens.

7.5 Gradientes em Imagens

Gradiente de um imagem é uma medida que, para cada região da imagem aponta a intensidade e a direção de um mudança de cor. Isso significa que o gradiente é um vetor, no sentido matemático da palavra. Para imagens que não tem mudança de cor o gradiente tem magnitude 0 e não tem direção. Já no exemplo cada "seta"representa o gradiente daquela região. Exemplo de imagem com gradientes disponível em https://commons.wikimedia.org/w/index.php?curid=1146042

7.6 Histograma de Gradientes orientados

O Histograma de Gradientes Orientados (HOG em inglês) é um descritor de características em imagens utilizado em visão computacional para o propósito de detecção de objetos. Essa técnica de conta a ocorrência de gradientes localizados em porções da imagem.

A ideia essencial por trás do Histograma de Gradientes Orientados é que a aparência do objeto e seu formato dentro da imagem pode ser descrito através da distribuição da intensidade de seus gradientes na direção das bordas. Para gerar um vetor de atributos que descrevem os formatos presentes na imagem o algoritmo divide a imagem em sub regiões chamadas células, geralmente regiões de 8x8, e para cada pixel em uma célula é calculada o histograma de gradientes. O resultado final é a concatenação dos gradientes de todas as células.

Com essas características é possível escrever um algoritmo capaz de responder se um determinado objeto está presente em uma imagem ou não.