PSI-5796: Processamento e Análise de Imagens e Vídeos
Primeiro semestre de 2020 2º exercício-programa Prof. Hae
Data de entrega: ~~31/05/2020~~ 07/06/2020 (domingo) até 24:00 horas

Obs. 1: Cada dia de atraso acarreta perda de 1 ponto no exercício.
Obs. 2: Este EP deve ser resolvido individualmente. EPs iguais receberão nota zero.

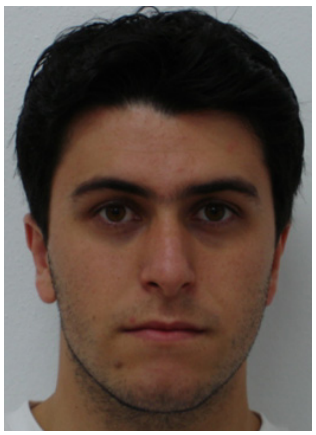
O objetivo deste exercício é identificar se um rosto é masculino ou feminino. O site abaixo contém vários bancos de dados de faces humanas:

http://fei.edu.br/~cet/facedatabase.html

Desse site, baixei os dois "ZIPs" abaixo e descompactei-os, resultando em 400 imagens JPG coloridas, com 360x260 pixels, de rostos frontais de expressão neutra (*a.jpg) e sorridente (*b.jpg), alinhadas manualmente.

frontalimages_manuallyaligned_part1 (~ 6MB)
frontalimages_manuallyaligned_part2 (~ 6MB)

Renomeei algumas dessas imagens, para que todas as imagens tenham nomes com 4 caracteres, por exemplo, 1a.jpg torne-se 001a.jpg. Dessas imagens, metade são rostos masculinos e metade são femininos.


095a.jpg


095b.jpg


096a.jpg


096b.jpg

Acrescentei três arquivos: treino.csv, valida.csv e teste.csv. O arquivo treino.csv contém os nomes de 200 imagens escolhidos aleatoriamente (100 masculinos e 100 femininos) a serem usadas durante o treinamento de aprendizagem de máquina, com rótulos 0=masculino e 1=feminino. Trecho desse arquivo:

```
002a.jpg;0
002b.jpg;0
012a.jpg;1
012b.jpg;1
...
```

Os arquivos valida.csv e teste.csv são semelhantes e contêm, cada um, os nomes de 100 imagens (50 masculinos e 50 femininos) a serem usadas respectivamente para validação e teste. O "zip" abaixo contém todos esses arquivos:

http://www.lps.usp.br/hae/psi3471/ep2-2019/frontalimages_manuallyaligned_with_csv.zip

Faça dois programas Python3 treino.py e teste.py. Treino.py deve fazer o treino (usando algum método de aprendizagem de máquina) utilizando as imagens listadas no arquivo treino.csv. Durante o treino, você pode utilizar as imagens listadas no arquivo valida.csv para verificar a taxa de erro do seu método. Você deve gravar o arquivo rede.h5 no diretório default com a rede treinada (uma ideia, não obrigatória, é gravar a rede com o menor erro de validação). Depois do treino, teste.py deve carregar rede.h5 e classificar as imagens teste.csv. Imprima a taxa de erro de teste e os nomes das imagens classificadas incorretamente. A forma de chamar programas deve ser **obrigatoriamente**:

```
$ python3 treino.py diretorio_do_banco_de_dados
(usa banco_de_dados para gerar rede.h5 no diretório default)
$ python3 teste.py diretorio_do_banco_de_dados
(lê rede.h5 do diretório default e classifica as imagens testes)
```

Exemplo:

```
windows> python treino.py c:\frontalImages
linux$ python3 treino.py ~/frontalImages
```

Os programas devem emitir uma mensagem amigável se no “diretorio_do_banco_de_dados” não tiver os arquivos csv ou as imagens necessárias para o processamento.

A saída de treino.py/treino.cpp deve ser algo como:

```
o menor erro de validacao: 3%
erro de treino: 0%
Tempo de processamento: 3 minutos.
```

A saída de teste.py/teste.cpp deve ser algo como:

```
erro de teste: 4%
Arquivos classificados incorretamente:
wwwa.jpg
xxxb.jpg
yyyb.jpg
zzza.jpg
```

O meu programa obteve taxa de erro de treino 0,1±0,4%, erro de validação 1,7±1,5% e erro de teste 5,9±2,6%, onde μ±σ% indica média μ e desvio-padrão σ de 20 testes. A cada execução de Tensorflow/Keras, uma taxa de erro diferente é obtida, devido à inicialização aleatória dos pesos e bias. Salvei a rede que apresentou o menor erro de validação durante o treino. O meu programa, escrito em Python/Tensorflow/Keras, levou 45 segundos para treinar usando GPU GTX960M. O mesmo programa levou 2 minutos para completar o treino usando CPU.

Obs. 1: Pode usar (se quiser) as bibliotecas OpenCV/Tensorflow/Keras.

Obs. 2: Entregue os programas-fontes (treino.py e teste.py), a rede obtida (rede.h5) e um documento PDF (relatorio.pdf) com os comentários descrevendo o funcionamento do programa. O envio do relatório é obrigatório (veja o anexo).

(a) Se você fez o programa no ambiente usado na classe (OpenCV/Tensorflow/Keras), basta entregar os arquivos acima para poder corrigir o seu programa.

(b) Se você quiser usar alguma biblioteca diferente ou programar em outro ambiente, converse antes com o professor. Neste caso, você deve combinar com o professor um horário para mostrar o seu programa funcionando.

Obs. 3: Compacte todos os arquivos como nome_sobrenome.zip e envie o arquivo para hae@lps.usp.br. Dentro do prazo, você pode substituir o arquivo anterior por um novo. Só o último arquivo entregue será corrigido.

Anexo: Relatórios dos exercícios programas

O mais importante numa comunicação escrita é que o leitor entenda, sem esforço e inequivocamente, o que o escritor quis dizer. O texto ficar "bonito" é um aspecto secundário. Se uma (pseudo) regra de escrita dificultar o entendimento do leitor, essa regra está indo contra a finalidade primária da comunicação. No site do governo americano [1], há regras denominadas de "plain language" para que comunicações governamentais sejam escritas de forma clara. As ideias por trás dessas regras podem ser usadas em outros domínios, como na escrita científica. Resumo abaixo algumas dessas ideias.

(1) Escreva para a sua audiência. No caso do relatório, a sua audiência será o professor ou o monitor que irá corrigir o seu exercício. Você deve enfocar na informação que o seu leitor quer conhecer. Não precisa escrever informações que são inúteis ou óbvias para o seu leitor.

(2) Organize a informação. Você é livre para organizar o relatório como achar melhor, porém sempre procurando facilitar o entendimento do leitor. Seja breve. Quebre o texto em seções com títulos claros. Use sentenças curtas. Elimine as frases e palavras que podem ser retiradas sem prejudicar o entendimento. Use sentenças em ordem direta (sujeito-verbo-predicado).

(3) Use palavras simples. Use o tempo verbal o mais simples possível. Evite cadeia longa de nomes, substituindo-os por verbos (em vez de "desenvolvimento de procedimento de proteção de segurança de trabalhadores de minas subterrâneas" escreva "desenvolvendo procedimentos para proteger a segurança dos trabalhadores em minas subterrâneas"). Minimize o uso de abreviações (para que o leitor não tenha que decorá-las). Use sempre o mesmo termo para se referir à mesma realidade (pode confundir o leitor se usar termos diferentes para se referir a uma mesma coisa). O relatório não é obra literária, não tem problema repetir várias vezes a mesma palavra.

(4) Use voz ativa. Deixe claro quem fez o quê. Se você utilizar oração com sujeito indeterminado ou na voz passiva, o leitor pode não entender quem foi o responsável (Ex: "Criou-se um novo algoritmo" - Quem criou? Você? Ou algum autor da literatura científica?). O site diz: "Passive voice obscures who is responsible for what and is one of the biggest problems with government writing."

(5) Use exemplos, diagramas, tabelas, figuras e listas. Ajudam bastante o entendimento.

O relatório deve conter pelo menos as seguintes informações:

Identificação

Nome, número USP, nome da disciplina, etc.

Breve enunciado do problema

Apesar do enunciado do problema ser conhecido ao professor/monitor, descreva brevemente o problema que está resolvendo. Isto tornará o documento compreensível para alguma pessoa que não tem o enunciado do EP à mão.

Técnica(s) utilizada(s) para resolver o problema

Descreva quais técnicas você usou para resolver o problema. Se você mesmo inventou a técnica, descreva a sua ideia, deixando claro que a ideia foi sua. Se você utilizou alguma técnica já conhecida, utilize o nome próprio da técnica (por exemplo, filtragem Gaussiana, algoritmo SIFT, etc.) juntamente com alguma referência bibliográfica onde a técnica está descrita. Use elementos gráficos como imagens intermediárias e diagramas, pois ajudam muito a compreensão. Não "copie-e-cole" código-fonte, a não ser que seja relevante. Use preferencialmente o pseudo-código.

Ambiente de desenvolvimento utilizado

1 https://plainlanguage.gov/guidelines/

Em qual plataforma você desenvolveu o programa? Como o professor/monitor pode compilar o programa? Você utilizou que bibliotecas?

Operação

Como o professor/monitor pode executar o programa? Que argumentos são necessários para a execução do programa? Há parâmetros que devem ser configurados? Quais arquivos de entrada são necessários? Quais arquivos de saída são gerados?

Resultados Obtidos

Descreva os resultados obtidos. Qual é o tempo de processamento típico? O problema foi resolvido de forma satisfatória?

Referências

Descreva o material externo utilizado, como livros/artigos consultados, websites visitados, etc.