

**PSI3472 Conceção e Implementação de Sistemas Eletrônicos Inteligentes**  
**Segundo Semestre de 2017**                      **2º exercício-programa**

Data de entrega: até 15/outubro/2017, domingo 24:00 horas.

Neste exercício, utilizaremos o banco de imagens CIFAR-10. Você deve baixar os 6 arquivos "\*.bin" (CIFAR-10 binary version) diretamente do site:

<http://www.cs.toronto.edu/%7Ekriz/cifar.html>

O banco de imagens CIFAR-10 consiste de 60.000 imagens coloridas 32x32 divididas em 10 classes, com 6.000 imagens por classe. Há 50.000 imagens de treino e 10.000 imagens de teste. O banco de imagens está dividido em 5 conjuntos de treino e 1 conjunto de teste, cada conjunto com 10.000 imagens. O conjunto de teste contém exatamente 1.000 imagens de cada classe selecionadas aleatoriamente. Os cinco conjuntos de treino contêm as imagens em ordem aleatória, mas alguns conjuntos podem conter mais imagens de uma classe que outras. Se somar as imagens dos cinco conjuntos de treino, dá exatamente 5.000 imagens de cada classe. Abaixo, as 10 classes do banco de imagens, assim como 10 imagens escolhidas aleatoriamente de cada classe.

0=airplane											
1=automobile											
2=bird											
3=cat											
4=deer											
5=dog											
6=frog											
7=horse											
8=ship											
9=truck											

Está incluído, na biblioteca "Tiny-DNN", o programa-exemplo train.cpp que classifica as imagens do CIFAR-10. Se você instalou Cekeikon, este programa encontra-se em:

(...)/cekeikon5/tiny\_dnn/examples/cifar10/train.cpp

No meu teste, esse programa atingiu taxa de erro de 30,56% após 30 epochs, usando parâmetros padrões. No meu computador (em Linux), cada epoch demorou aproximadamente 127s. O site do CIFAR-10 diz que é possível chegar a 18% de erro sem aumentar artificialmente dados de treino e 11% com aumento, usando rede neural convolucional. A menor taxa de erro obtida parece ser 3,47%: [http://rodrigob.github.io/are\\_we\\_there\\_yet/build/classification\\_datasets\\_results.html](http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html)

O objetivo deste exercício é classificar as imagens-testes de CIFAR-10 usando diferentes métodos de aprendizagem de máquina. Faça os programas em C/C++, usando (opcionalmente) bibliotecas OpenCV, Cekeikon e Tiny-DNN. Teste (pelo menos) os seguintes métodos, sempre ajustando os parâmetros para minimizar o erro.

Passo intermediário A) Compile e execute o programa:

```
(...)/cekeikon5/tiny_dnn/examples/cifar10/train.cpp
```

Passo intermediário B) Faça um programa leitura.cpp que lê um arquivo \*.bin de Cifar10 e imprime a primeira imagem do banco de dados como \*.png.

```
>leitura test_batch.bin saida.png
```

1) Vizinho mais próximo (k-NN). Você pode programar a sua própria função de busca dos vizinhos mais próximos ou usar as classes CvKNearest ou FlaNN do OpenCV. A chamada do programa deve ser como abaixo, onde cifar10 é o diretório onde estão localizados os arquivos do banco de dados CIFAR10.

```
>knn /home/usuario/cifar10
>knn c:\base\cifar10
```

Neste e no próximo item, os programas devem responder algo como:

```
>Taxa de erro: ??.%
>Tempo de treino: ????.?s.
>Tempo de teste: ????.?s.
```

2 - opcional) Um dos métodos de aprendizagem de máquina do OpenCV diferente de vizinho mais próximo, por exemplo, aprendizagem Bayesiana, árvore de decisão, boosting, rede neural (a classe CvANN\_MLP do OpenCV), SVM (support vector machine), etc. Nota: Este item é opcional por que estas técnicas não foram vistas nesta disciplina. A chamada deve ser:

```
>mle /home/usuario/cifar10
>mle c:\base\cifar10
```

3) Rede neural convencional (rede não-convolucional, onde todas as camadas são do tipo "fully-connected") da biblioteca Tiny-DNN. Você pode escolher a estrutura da rede, a função de ativação e os parâmetros para minimizar o erro. Grave no arquivo "fulcon.net" o modelo da melhor rede obtida. A chamada deve ser:

```
>fulcon /home/usuario/cifar10
>fulcon c:\base\cifar10
```

Neste e no próximo item, os programas devem responder algo como:

```
>Taxa de erro após epoch 1: ??.%
>Taxa de erro após epoch 2: ??.%
>Taxa de erro após epoch 3: ??.%
>Menor taxa de erro: ??.%
>Tempo de treino: ????.?s.
>Tempo de teste: ????.?s.
```

4) Rede neural profunda convolucional. Procure minimizar o erro, para atingir taxa de erro menor que 30% do programa-exemplo. Grave no arquivo "convol.net" o modelo da melhor rede obtida. A chamada deve ser:

```
>convol /home/usuario/cifar10
>convol c:\base\cifar10
```

Antes de aplicar os métodos acima, você pode (opcionalmente) executar qualquer pré-processamento para diminuir a taxa de erro. Por exemplo, pode (entre outras possibilidades): diminuir ou aumentar a resolução das imagens, aumentar artificialmente o número de imagens de treinamento, equalizar o histograma, mudar o espaço das cores, PCA (principal component

analysis), HOG (histogram of oriented gradient), LBP (local binary pattern), aplicar filtros lineares, etc.

Você deve entregar os programas que você fez, os modelos das duas redes (fulcon.net e convol.net) e um relatório ("diário de bordo") descrevendo os passos que seguiu. Compacte todos os arquivos com os nomes do grupo (nome1\_sobrenome1\_nome2\_sobrenome2.zip) e envie via edisciplinas (Moodle).

## **Anexo: Relatórios dos exercícios programas**

A primeira regra para uma comunicação clara é escrever para a sua audiência (<http://www.plainlanguage.gov/howto/guidelines/FederalPLGuidelines/think.cfm>). No caso do relatório, a sua audiência será o professor ou o monitor que irá corrigir o seu exercício. Assim, você não precisa escrever informações que são óbvias para professor/monitor. Por outro lado, precisa escrever aqueles dados que ajudarão professor/monitor a avaliar corretamente a sua solução. Seja breve. Passe as informações necessárias usando menos palavras/páginas possível.

### **Identificação**

Seu nome, número USP, nome da disciplina, etc.

**Para cada um dos sub-programas, escreva:**

#### **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 em português a sua ideia, deixando claro que a ideia foi sua. 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? <http://www.plainlanguage.gov/howto/guidelines/FederalPLGuidelines/writeActive.cfm>). 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**

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.