República Federativa do Brasil
Ministério da Educação
Universidade Federal do Amazonas
Instituto de Computação

Tópicos Especiais em Aprendizado de Máquina e Mineração de Dados

Trabalho Prático

Professores: *Marco Cristo* e *Eulanda Miranda*

Alunos:

Reidentificação de pessoas em imagens de câmeras de segurança

Neste trabalho, nosso objetivo é re-identificar pessoas capturadas por câmeras de segurança. Em particular, dadas duas imagens capturadas por diferentes câmeras, queremos determinar se a pessoa observada nas imagens é a mesma.

Na figura abaixo temos exemplos de dez pares de imagens obtidas pelas câmeras, rotuladas como iguais e diferentes, dependendo se elas capturaram a mesma pessoa ou não:

O que deve ser feito

Você deve resolver este problema usando três modelos de redes neurais, conforme descrito a seguir:

- **S1**: Rede Siamesa, onde modelo base é uma CNN;
- **S2**: Rede Siamesa, onde modelo base é uma CNN combinada com um Autocodificador;
- **S3**: Rede Siamesa, onde modelo base é uma CNN combinada com um Autocodificador, com pré-treino do autocodificador;

Os alunos podem optar por quaisquer arquiteturas que quiserem de CNN e Autocodificador. Contudo, os resultados de referência deste trabalho foram obtidos com as seguintes arquiteturas:

- **S1**:
  - 8 convolução 2D kernel 3x3, BatchNormalization, Relu, Pooling 2x2;
  - 16 convolução 2D kernel 3x3, BatchNormalization, Relu, Pooling 2x2;
  - 32 convolução 2D kernel 3x3, BatchNormalization, Relu, Pooling 2x2;
  - 64 convolução 2D kernel 3x3, BatchNormalization, Relu, Pooling 2x2;
  - Embedding de 64 é usado para Siamesa

- **S2** e **S3**:
  - 8 convolução 2D kernel 3x3, BatchNormalization, Relu, Pooling 2x2;
  - 16 convolução 2D kernel 3x3, BatchNormalization, Relu, Pooling 2x2;
  - 32 convolução 2D kernel 3x3, BatchNormalization, Relu, Pooling 2x2;
  - 64 convolução 2D kernel 3x3, BatchNormalization, Relu, Pooling 2x2;
  - Densa de 64 é usada para conectar modelos
  - Codificador com Densa de 64, Densa de 32, Densa de 16 (espaço latente);
  - Decodificador com entrada de 16, Densa de de 32, Densa de 64
  - Embedding de 64 é usado para Siamesa

Em 100 épocas, considerando os mesmos conjuntos de treino e validação disponível com este trabalho, estas redes alcançaram taxas de acurácia entre 70% e 80%, dependendo de diferentes configurações de parâmetros usadas.

Datasets & Acesso a GPU

Dataset **airport-alunos**:

- **Treino**: 200 pessoas vistas nas câmeras 1 e 2. Para cada pessoa, há uma variedade de imagens que correspondem a frames do vídeo capturado. Como resultado, o treino é formado de 14094 imagens da câmera 1 e 14939 da câmera 2.
- **Validação**: 50 pessoas diferentes vistas nas câmeras 1 e 2. Considerando o total de frames por pessoa, a validação é formada por 2824 imagens da câmera 1 e 3648 da câmera 2.

Os dataset está disponível em:

- Google Drive: https://drive.google.com/file/d/14CB3Vw4jPf-8-DAriB9XDq4mfbs4o747/view?usp=sharing (https://drive.google.com/file/d/14CB3Vw4jPf-8-DAriB9XDq4mfbs4o747/view?usp=sharing)
- UFAM: http://papaleguas.icomp.ufam.edu.br/~marco/downloads/airport-alunos.tgz (http://papaleguas.icomp.ufam.edu.br/~marco/downloads/airport-alunos.tgz)

Para ter acesso a uma GPU, é recomendável usar o Google Colaboratory. Se você tem conta no Google, basta acessar o Google Drive e criar um novo notebook no Colaboratory. Este vai ser criado em uma VM que pode ser configurada para um runtime baseado em Python 3 e aceleração com GPU (Notebook settings). Neste caso, o dataset acima deveria ser baixado *diretamente* do GDrive para o notebook. Para isso, supondo que você quer baixar e descompactar em um sub-diretório `datasets` em sua VM no Colaboratory, use os comandos:

```
!pip install gdown
!gdown https://drive.google.com/uc?id=14CB3Vw4jPf-8-DAriB9XDq4mfbs4o747
!mkdir datasets
!tar -C datasets -xzf airport-alunos.tar.gz
```

As imagens do dataset estão dispostas conforme a estrutura de diretórios abaixo:

```
airport-alunos
├── treino
│   ├── 0
│   │   └── 0 a 199: imagens de pessoas 0 a 199 obtidas pela câmera 1
│   └── 1
│       └── 0 a 199: imagens de pessoas 0 a 199 obtidas pela câmera 2
└── val
    ├── 0
    │   └── 200 a 249: imagens de pessoas 200 a 249 obtidas pela câmera 1
    └── 1
        └── 200 a 249: imagens de pessoas 200 a 249 obtidas pela câmera 2
```

Você pode obter a lista completa dos arquivos de imagens com os comandos:

```
file_names_treino = !find datasets/airport-alunos/treino -name '???.png' | sort
file_names_val = !find datasets/airport-alunos/val -name '???.png' | sort
```

Uma vez que os datasets estão disponíveis, você deve criar batches de pares de imagens. Para garantir balanceamento, devem ser criados o mesmo número de pares de imagens iguais e distintas. As funções abaixo criam pares desta forma em memória (cortesia do Mikael):

```
def person_to_img(file_names):
    cam_path_to_img = {}
    pids = [[], []]
    img_counts = [0, 0]
    for f in tqdm(file_names):
        cid = int(f.split('/')[-3])
        pid = int(f.split('/')[-2])
        iid = int(f.split('/')[-1][:3])
        pids[cid].append((pid, iid))
        cam_path_to_img[cid, pid, iid] = img_to_array(load_img(f))
        img_counts[cid] += 1
    print('Images in cam1 = %d, cam2 = %d'%(img_counts[0], img_counts[1]))
    # dic person_to_img e person ids em cada camera
    return cam_path_to_img, list(set(pids[0])), list(set(pids[1]))

def filter_possible(pid1, pids2, same=True):
    person_number = pid1[0]
    if same is True:
        possible_ids2 = [pid2 for pid2 in pids2 if pid2[0] == person_numbe
r]
    else:
        possible_ids2 = [pid2 for pid2 in pids2 if pid2[0] != person_numbe
r]
    return possible_ids2

def combine_cam_files(pids1, pids2):
    i = 0
    d = {}
    for pid1 in tqdm(pids1):
        if i % 2 == 0:
            d[pid1] = filter_possible(pid1, pids2, same=True)
        else:
            d[pid1] = filter_possible(pid1, pids2, same=False)
        i += 1
    return d

def get_batch(d_combination, cam_img_dict):
    pairs_labels = []
    i = 0

    for key, value in d_combination.items():
        if i % 2 == 0:
            t = ((0, *key), (1, *random.choice(value)), 1.)
        else:
            t = ((0, *key), (1, *random.choice(value)), 0.)
        i += 1
        pairs_labels.append(t)

    train_data = pairs_labels
    X_a_train = np.array([cam_img_dict[x[0]] for x in train_data], dtype =
np.float32)
    X_b_train = np.array([cam_img_dict[x[1]] for x in train_data], dtype =
```

```
np.float32)
    y_train = np.array([x[2] for x in train_data], dtype = np.float32)

    return X_a_train, X_b_train, y_train
```

Com estas funções, para se obter batches de validação, você usaria os comandos:

```
# carrega imagens para memória
cam_img_dict_val, pids1_val, pids2_val = person_to_img(file_names_val)

# gera listas de similares e dissimilares na memória
d_combination_val = combine_cam_files(pids1_val, pids2_val)

# obtem pares de batches apartir das imagens em memória
X_a_val, X_b_val, y_val = get_batch(d_combination_val, cam_img_dict_val)
```

Neste caso, `X_a_val` e `X_b_val` correspondem aos pares de imagens a serem dadas para a Rede. O vetor `y_val` indica se os pares são iguais ou diferentes. Para plotar alguns exemplos destes pares de imagens, você pode usar as funções:

```
# plot pics side by side
def show_side_by_side(figs, titles = None, limit = 10,
                      figsize=(20, 4), cmap = 'gray', grid = False):
    minval = min(limit, figs.shape[0])
    plt.figure(figsize = figsize)
    for i in range(minval):
        subplot = plt.subplot(1, limit, i + 1)
        extent = (0, figs[i].shape[1], figs[i].shape[0], 0)
        subplot.imshow(figs[i], cmap = cmap, extent = extent)
        if titles:
            subplot.set_title(titles[i])

        if grid:
            subplot.grid(color='gray', linestyle='-', linewidth=1)
        else:
            subplot.get_xaxis().set_visible(False)
            subplot.get_yaxis().set_visible(False)

    plt.show()

def plot_sample(cam1, cam2, val):
    rindices = np.random.choice(np.array(range(cam1.shape[0])), 10, replac
e=False)
    show_side_by_side(cam1[rindices], titles = [('DIF' if n == 0 else 'IGU
AL') for n in val[rindices]])
    show_side_by_side(cam2[rindices])

plot_sample(X_a_val/255, X_b_val/255, y_val)
```

Avaliação

- Alunos devem devolver notebook com os modelos finais usados e os resultados obtidos nos dados de validação;
- Modelos serão avaliados em conjunto separado de dados;
- 20% da nota será baseada em ranking na turma;
  - A posição dos alunos no ranking será definida pelos scores obtidos pelos modelos considerando apenas diferenças significativas;
- 80% da nota será referente à avaliação dos modelos e resultados;
- As métricas a serem consideradas na avaliação são a acurácia e a curva de *loss* ao longo de 100 épocas;
- Data de entrega: **7 de novembro**

Observações

- Os trabalho devem ser realizados por um ou dois alunos, sem exceções;
- As implementações devem ser feitas em tensorflow 2. No caso do colaboratory, você deve verificar como fazer isso. No meu caso, tive que executar no início do notebook os comandos:
  - `%tensorflow_version 2.x`
  - `pip install -U gast==0.2.2`
- Caso você tenha problemas com *overfitting*, você é livre para tentar diferentes estratégias de atenuação, tais como:
  - Redução do tamanho dos batches;
  - Data augmentation;
  - Regularização, em particular, Dropout e Batch Normalization;