# AI
# 演習資料

# 転移学習の演習

- 背景画像と 2 人の顔画像を使って 3 分類の画像分類を行います。
- 学習には Colaboratory を使い、予測はノート PC のローカル環境で行います。
- ローカルの実行環境については、後ほど行う学習済みモデルの利用のために、ベータ版ですが TensorFlow2.0.0-beta1 を使用します。

## ■学習用画像データセット作成

「画像の水増し.py」、「学習用データセット作成.py」を参照。ローカル環境で実行することを想定していますが、Colaboratory でも実行できます。

1. 各分類の画像を数枚ずつ用意する。
2. Image Augmentation により分類ラベルごとに画像を水増しする（画像の水増し.py）。
   - デフォルトでは「FaceImage」フォルダに画像を置く。
   - 後で生成した画像の一部を学習評価用データにすることを考慮して枚数を決める。
   - 入力画像のサイズはここで決定する。
     - ✓ 100x100 を目安に。
   - 生成された画像は「AugImgs.npy」という名前で一つの npy ファイルとして出力される。
     - ✓ 分類ラベル毎に出力ファイル名を変更する。
3. 全画像データを統合して一つの npy ファイルにする（学習用データセット作成.py）。
   - 先に作成した各分類の npy ファイルを一つの npy ファイルに統合する。
   - 統合する際にラベルデータを一緒に作成する。
   - デフォルトでは各分類の 20%を学習評価用データとしている。
   - 学習用データセット
     - ✓ training_imgs.npy（入力画像データ）
     - ✓ training_labels.npy（正解ラベルデータ）

- 学習評価用データセット
  - ✓ validation_imgs.npy（入力画像データ）
  - ✓ validation_labels.npy（正解ラベルデータ）

## ■シンプルな CNN による分類モデルの作成

「CSAJ2019Nov 顔認識 CNN.ipynb」を参照。まずはシンプルな CNN で分類モデルを作ってみます。Colaboratory で実行します。

1. 作成した学習用データセット、学習評価用データセットを GoogleDrive のフォルダに置く。
2. 「CSAJ2019Nov 顔認識 CNN.ipynb」の data_dir を修正し、データを置いたフォルダへのパスを通す。
3. NN の定義部分を修正してニューラルネットワークを構築し分類モデルを学習させる。
4. デフォルトではデータを置いたフォルダに「CNN.h5」というファイルが作成される。

## ■作成した分類モデルによる画像分類

「detection_roi.py」を参照。ローカル環境で実行します。

1. 作成した学習モデルのファイル（デフォルトでは CNN.h5）をソースファイルと同じフォルダにダウンロードする。
2. ソースコード内の「img_rows」と「img_cols」の値を、学習データの画像サイズと一致させる。
3. 実行するとノート PC のカメラ映像が表示される。カメラ映像の中の枠内が分類の対象となる。
4. 分類結果によって枠の色が変わります。

※背景と顔は比較的正確に分類できます。しかし、演習内で試すことができる学習時間（10 分以内程度）で 2 人のうちどちらの顔かの分類が可能になる程度の学習モデル生成は難しいかもしれません。

## ■転移学習による分類モデル作成

転移学習の概要についてはスライド資料参照。

学習用データの作成は、先の顔分類モデル演習と同じ要領です。学習済みモデルで使用する画像サイズはモデルによって決まっていますが、モデルを呼び出すときに画像サイズを指定することで任意の画像サイズを使うことができます。

「CSAJ2019Nov 転移学習.ipynb」参照

### 12～46 行目 学習用データ読み込みや前処理など

```
---
data_dir = "./drive/My Drive/CSAJ_AI2019Nov/Data/Transfer/"

train_images = np.load(data_dir+"training_imgs100x100.npy")
train_labels = np.load(data_dir+"training_labels100x100.npy")
valid_images = np.load(data_dir+"validation_imgs100x100.npy")
valid_labels = np.load(data_dir+"validation_labels100x100.npy")

IMG_SIZE = 100
IMG_SHAPE = (IMG_SIZE, IMG_SIZE, 3)

# 画像の解像度とクラス数
img_rows = IMG_SIZE
img_cols = IMG_SIZE
num_classes = 3
num_channels = 3


# データ数，行数，列数，3（色チャンネル数）の 4 次元のテンソルに変形
train_images = train_images.reshape(train_images.shape[0],img_rows,
img_cols, num_channels)
valid_images = valid_images.reshape(valid_images.shape[0],img_rows,
img_cols, num_channels)


# データ形式をそろえる
train_images = train_images.astype('float32')
valid_images = valid_images.astype('float32')
train_labels = train_labels.astype('int32')
valid_labels = valid_labels.astype('int32')


# 正解データは整数値になっているので，One-hot ベクトルに変換
train_labels = tf.keras.utils.to_categorical(train_labels,num_classes)
valid_labels = tf.keras.utils.to_categorical(valid_labels,num_classes)
```

```
#画像の画素値を MobileNet V2 用に正規化
train_images =
tf.keras.applications.mobilenet_v2.preprocess_input(train_images)
valid_images =
tf.keras.applications.mobilenet_v2.preprocess_input(valid_images)
---
```

**51 行目 学習済みモデル読み込み**

```
---
base_model =
tf.keras.applications.mobilenet_v2.MobileNetV2(input_shape=IMG_SHAPE,
                                               include_top=False,
                                               weights='imagenet')
# base_model の重みは変更しない
base_model.trainable = False
---
```

**62～67 行目 読み込んだ学習済みモデルに分類器にあたる層を追加**

```
---
x = base_model.output
x = tf.keras.layers.GlobalAveragePooling2D()(x)
predictions = tf.keras.layers.Dense(num_classes,
activation=tf.nn.softmax)(x)

# ネットワーク定義
model = tf.keras.Model(inputs = base_model.input, outputs = predictions)

# ニューラルネットワークの構造を表示
model.summary()
---
```

**73～86 行目 モデル構築と学習**

```
---
base_learning_rate = 0.0001
model.compile(optimizer=tf.keras.optimizers.RMSprop(lr=base_learning_rate
),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

#一度に学習するバッチサイズ
batchsize=100

# 学習
history = model.fit(train_images, train_labels, epochs=30,
batch_size=batchsize,
                    validation_data=(valid_images,valid_labels),
verbose=1)

# TensorFlow 2 系は compile 情報も含めてファイルに保存する.
model.save(data_dir+"transfer_imagenet.h5")
```

keras で利用できる ImageNet（http://www.image-net.org）のデータによる学習済みモデルは次のとおりです。（https://keras.io/ja/applications/ より）

- Xception (François Chollet, Google, keras の作者)
- VGG16 (Visual Geometry Group, Univ. Of Oxford)
- VGG19 (Visual Geometry Group, Univ. Of Oxford)
- ResNet50 (Kaiming He, 当時 Microsoft Research Asia, 現 Facebook AI Research)
- InceptionV3 (Christian Szegedy, Google)
- InceptionResNetV2 (Christian Szegedy, Google)
- MobileNet (Andrew Howard, Google)
- DenseNet (Gao Huang, 精華大学)
- NASNet (Barret Zoph, Google Brain)
- MobileNetV2 (Mark Sandler, Google)

以上。

# ローカル PC における実装環境構築手順

- グラフィックスボードのドライバを最新にする
- CUDA（TensorFlow のバージョンに合わせたもの）のインストール
- cuDNN（CUDA のバージョンに合わせる）のインストール

> ここまでの部分は、GPU 版の TensorFlow（tensorflow-gpu）を使う場合です。
> CUDA、cuDNN、と TensorFlow のバージョン合わせには注意してください。
> 今回は CPU 版を使うので必要ありません。

- Miniconda のインストール
  以降は Miniconda のプロンプトから作業を行う

- Python3.7 の仮想環境構築 （tf2_py37 は仮想環境名、変更可）

```
> conda create -n tf2_py37 python=3.7
```

- 仮想環境の起動

```
> activate tf_py37
```

- pip のアップグレード

```
> python -m pip install --upgrade pip
```

- Matplotlib のインストール

```
> pip install matplotlib
```

- OpenCV のインストール

```
> pip install opencv-python
```

- Pandas のインストール

```
> pip install pandas
```

- SciPy のインストール

```
> pip install scipy
```

- TensorFlow のインストール（今回は 2.0.0-beta1 をインストールする）

```
> pip install tensorflow==2.0.0b1
```

# PyCharm 操作

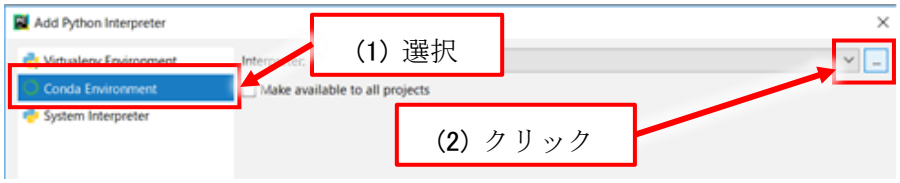
**プロジェクト新規作成**

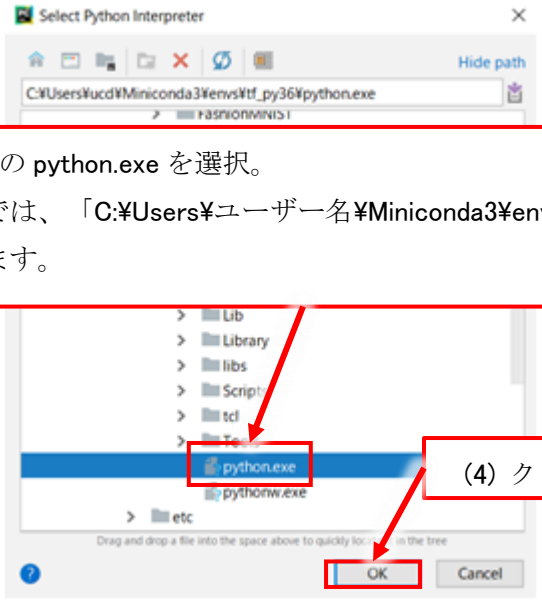
「File」から「New Project」

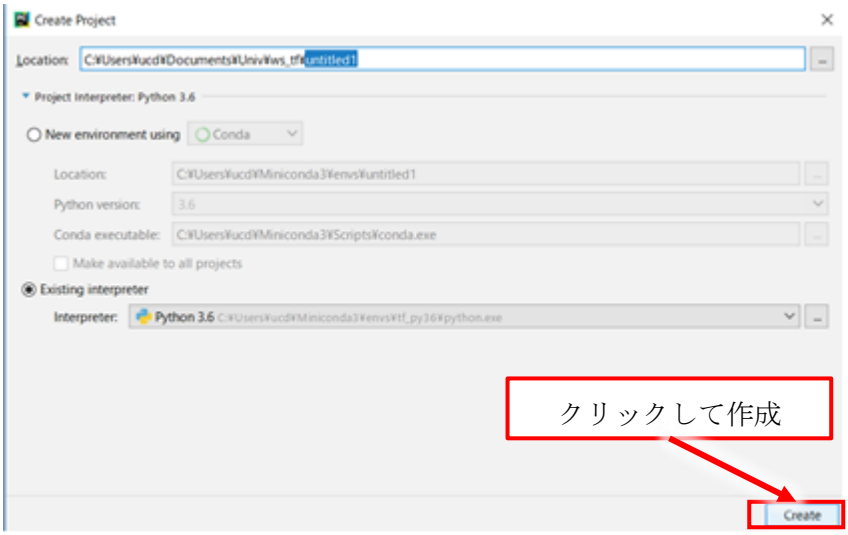
**プロジェクトの設定（Python インタープリタの指定）**

※Miniconda で作成した仮想環境の Python インタープリタを指定します。


conda の仮想環境を指定



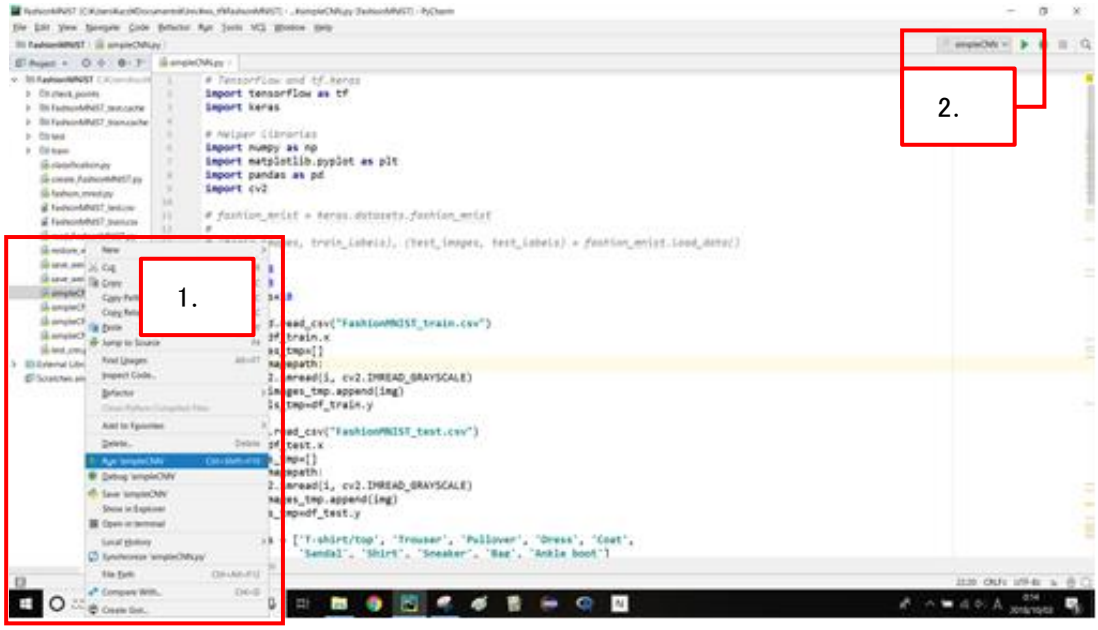



**プログラムの実行方法**

※ファイルは全てオートセーブなので「保存」メニューはありません。

実行方法は複数あります。

1. 実行したいソースファイル名の上で右クリックし、「Run ファイル名」をクリックする
2. 実行したいソースファイルを選択して実行ボタンをクリックする




以上。