Project : to Identify Facial Keypoint

- Here we're defining and training a convolutional neural network to perform facial keypoint detection, and using computer vision techniques to transform images of faces. The first step in any challenge like this will be to load and visualize the data you'll be working with.

- Facial keypoints (also called facial landmarks) are the small magenta dots shown on each of the faces in the image above. In each training and test image, there is a single face and **68 keypoints, with coordinates (x, y), for that face**. These keypoints mark important areas of the face: the eyes, corners of the mouth, the nose, etc

- ## Data Preprocessing
- First we define a function **show_keypoints** that takes in an image and keypoints and displays them. **note that these images are not all of the same size**, and neither are the faces! To eventually train a neural network on these images, we'll need to standardize their shape.

- ## Dataset class and Transformations
- To prepare our data for training, we'll be using PyTorch's Dataset class. Much of this this code is a modified version of what can be found in the PyTorch data loading tutorial.

- ### *Dataset class*
- torch.utils.data.Dataset is an abstract class representing a dataset. This class will allow us to load batches of image/keypoint data, and uniformly apply transformations to our data, such as rescaling and normalizing images for training a neural network

- ## Transforms
- Now, the images above are not of the same size, and neural networks often expect images that are standardized; a fixed size, with a normalized range for color ranges and coordinates, and (for PyTorch) converted from numpy lists and arrays to Tensors.

- Therefore, we will need to write some pre-processing code. Let's create four transforms:

- Normalize: to convert a color image to grayscale values with a range of [0,1] and normalize the keypoints to be in a range of about [-1, 1]
- Rescale: to rescale an image to a desired size.
- RandomCrop: to crop an image randomly.
- ToTensor: to convert numpy images to torch images.

- ## Test out the transforms
- Let's test these transforms out to make sure they behave as expected. As you look at each transform, note that, in this case, **order does matter**. For example, you cannot crop a image using a value smaller than the original image (and the orginal images vary in size!), but, if you first rescale the original image, you can then crop it to any size smaller than the rescaled size.

- ## Create the transformed dataset
- Apply the transforms in order to get grayscale images of the same shape. Verify that your transform works by printing out the shape of the resulting data (printing out a few examples should show you a consistent tensor size).

- ## Data Iteration and Batching
- Right now, we are iterating over this data using a for loop, but we are missing out on a lot of PyTorch's dataset capabilities, specifically the abilities to:

- Batch the data
- Shuffle the data
- Load the data in parallel using multiprocessing workers.
- **torch.utils.data.DataLoader** is an iterator which provides all these features, and we'll see this in use in the notebook, when we load data in batches to train a neural network!

- ## Define the Convolutional Neural Network
- After you've looked at the data you're working with and, in this case, know the shapes of the images and of the keypoints, you are ready to define a convolutional neural network that can *learn* from this data

- ## Recall that CNN's are defined by a few types of layers:

- Convolutional layers
- Maxpooling layers
- Fully-connected layers