UDACITY

PROJECT SPECIFICATION

Facial Keypoint Detection

Files Submitted

CRITERIA	MEETS SPECIFICATIONS
Submission Files	The submission includes **models.py** and the following Jupyter notebooks, where all questions have been answered and training and visualization cells have been executed: **2. Define the Network Architecture.ipynb**, and **3. Facial Keypoint Detection, Complete Pipeline.ipynb**. Other files may be included, but are not necessary for grading purposes. Note that all your files will be zipped and uploaded should you submit via the provided workspace.

`models.py`

CRITERIA	MEETS SPECIFICATIONS

CRITERIA	MEETS SPECIFICATIONS
Define a CNN in `models.py`	Define a convolutional neural network with at least one convolutional layer, i.e. `self.conv1 = nn.Conv2d(1, 32, 5)`. The network should take in a grayscale, square image.

Notebook 2: Define the Network Architecture

CRITERIA	MEETS SPECIFICATIONS
Define the `data_transform` for training and test data	Define a `data_transform` and apply it whenever you instantiate a DataLoader. The composed transform should include: rescaling/cropping, normalization, and turning input images into torch Tensors. The transform should turn any input image into a normalized, square, grayscale image and then a Tensor for your model to take it as input.
Define the loss and optimization functions	Select a loss function and optimizer for training the model. The loss and optimization functions should be appropriate for keypoint detection, which is a regression problem.
Train the CNN	Train your CNN after defining its loss and optimization functions. You are encouraged, but not required, to visualize the loss over time/epochs by printing it out occasionally and/or plotting the loss over time. Save your best trained model.

CRITERIA	MEETS SPECIFICATIONS
Answer questions about model architecture	After training, all 3 questions about model architecture, choice of loss function, and choice of batch_size and epoch parameters are answered.
Visualize one or more learned feature maps	Your CNN "learns" (updates the weights in its convolutional layers) to recognize features and this criteria requires that you extract at least one convolutional filter from your trained model, apply it to an image, and see what effect this filter has on an image.
Answer question about feature visualization	After visualizing a feature map, answer: what do you think it detects? This answer should be informed by how a filtered image (from the criteria above) looks.

Notebook 3: Facial Keypoint Detection

CRITERIA	MEETS SPECIFICATIONS
Detect faces in a given image	Use a Haar cascade face detector to detect faces in a given image.
Transform each detected face into an input Tensor	You should transform any face into a normalized, square, grayscale image and then a Tensor for your model to take in as input (similar to what the `data_transform` did in Notebook 2).

CRITERIA	MEETS SPECIFICATIONS
Predict and display the keypoints on each face	After face detection with a Haar cascade and face pre-processing, apply your trained model to each detected face, and display the predicted keypoints for each face in the image.

Suggestions to Make Your Project Stand Out!

- Initialize the weights of your CNN by sampling a normal distribution or by performing Xavier initialization so that a particular input signal does not get too big or too small as the network trains.
- In Notebook 4, create face filters that add sunglasses, mustaches, or any .png of your choice to a given face in the correct location.
- Use the keypoints around a person's mouth to estimate the curvature of their mouth and create a smile recognition algorithm .
- Use OpenCV's k-means clustering algorithm to extract the most common facial poses (left, middle, or right-facing, etc.).
- Use the locations of keypoints on two faces to swap those faces.
- Add a rotation transform to our list of transformations and use it to do data augmentation.