

[< Return to Classroom](#)

Facial Keypoint Detection

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

You have done a fantastic job completing the facial keypoint detection project. The facial keypoint detection is a well known [machine learning challenge](#). Here are some of my suggestion which can help in improving accuracy for keypoint detection:

1. You can use transfer learning.
2. You can use some deeper network architecture such as inception network.

You might also want to check the resource below to further improve your model.

1. [Facial keypoints detection using Neural Network](#).
2. [Facial Keypoints Detection: An Effort to Top the Kaggle Leaderboard](#).

Keep up the great work.
Stay safe, stay healthy!

Files Submitted

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.

Great, You have submitted all the required files.

`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.

Awesome- Well done, the use of more than 2 convolution layers, dropout and max pooling layers is much appreciated, it helps in detecting complex features as well as reducing overfitting.

Suggestion- You might want to try transfer learning to improve the accuracy. You can find tutorials for the transfer learning [here](#).

Notebook 2: Define the Network Architecture

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.

Depending on the complexity of the network you define, and other hyperparameters the model can take some time to train. We encourage you to start with a simple network with only 2 layers. You'll be graded based on the implementation of your models rather than accuracy.

Awesome - Good job on using the composed transform that rescales, random crop, normalizes and turns the images into torch Tensors.

Suggestion - To learn about the other available transformations you can look at its [official documentation](#).

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.

Awesome - A good regression loss and optimizer has been chosen.

Suggestion - You can also try the combination of smooth L1 loss and adam optimizer. Smooth L1 loss can be interpreted as combination of L1-loss and L2-loss. It also helps in reducing the effect of outliers. Also the adam optimizer is widely preferred with convolutional neural networks.

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.

Awesome- Good job training the network and saving the model checkpoint. Also great job displaying the loss over epochs.

After training, all 3 questions about model architecture, choice of loss function, and choice of batch_size and epoch parameters are answered.

Awesome- The answers to the 3 questions are quite analytical in nature and your reasoning is clear and sound, well done!

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.

Awesome- Good job using filter2D function provided by OpenCV. The CNN does learn to recognize features in the image.

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.

Nicely done identifying the filter as a filter that detects the edges in the image.

Notebook 3: Facial Keypoint Detection

Use a Haar cascade face detector to detect faces in a given image.

Awesome - Haar cascade face detector is correctly used to detect faces ,
Suggestion - You may also want to try some alternate face detectors present [here](#).

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).

Awesome- Good job on performing the following steps:

1. Convert the face from RGB to grayscale
2. Normalize the grayscale image so that its color range falls in [0,1] instead of [0,255]
3. Rescale the detected face to be the expected square size for your CNN
4. Reshape the numpy image into a torch image.

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.

Awesome- Nice job while displaying the facial keypoints on each face in the image.

 [DOWNLOAD PROJECT](#)

[RETURN TO PATH](#)

Rate this review

START