2. Define the Network Architecture

May 17, 2020

0.1 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.

In this notebook and in `models.py`, you will: 1. Define a CNN with images as input and keypoints as output 2. Construct the transformed FaceKeypointsDataset, just as before 3. Train the CNN on the training data, tracking loss 4. See how the trained model performs on test data 5. If necessary, modify the CNN structure and model hyperparameters, so that it performs *well* *

* What does *well* mean?

"Well" means that the model's loss decreases during training **and**, when applied to test image data, the model produces keypoints that closely match the true keypoints of each face. And you'll see examples of this later in the notebook.

0.2 CNN Architecture

Recall that CNN's are defined by a few types of layers: * Convolutional layers * Maxpooling layers * Fully-connected layers

You are required to use the above layers and encouraged to add multiple convolutional layers and things like dropout layers that may prevent overfitting. You are also encouraged to look at literature on keypoint detection, such as this paper, to help you determine the structure of your network.

0.2.1 TODO: Define your model in the provided file `models.py` file

This file is mostly empty but contains the expected name and some TODO's for creating your model.

0.3 PyTorch Neural Nets

To define a neural network in PyTorch, you define the layers of a model in the function `__init__` and define the feedforward behavior of a network that employs those initialized layers in the function `forward`, which takes in an input image tensor, `x`. The structure of this Net class is shown below and left for you to fill in.

Note: During training, PyTorch will be able to perform backpropagation by keeping track of the network's feedforward behavior and using autograd to calculate the update to the weights in the network.

Define the Layers in `__init__` As a reminder, a conv/pool layer may be defined like this (in `__init__`):

```
# 1 input image channel (for grayscale images), 32 output channels/feature maps, 3x3 square conv
self.conv1 = nn.Conv2d(1, 32, 3)

# maxpool that uses a square window of kernel_size=2, stride=2
self.pool = nn.MaxPool2d(2, 2)
```

Refer to Layers in `forward` Then referred to in the `forward` function like this, in which the conv1 layer has a ReLu activation applied to it before maxpooling is applied:

```
x = self.pool(F.relu(self.conv1(x)))
```

Best practice is to place any layers whose weights will change during the training process in `__init__` and refer to them in the `forward` function; any layers or functions that always behave in the same way, such as a pre-defined activation function, should appear *only* in the `forward` function.

Why models.py You are tasked with defining the network in the `models.py` file so that any models you define can be saved and loaded by name in different notebooks in this project directory. For example, by defining a CNN class called `Net` in `models.py`, you can then create that same architecture in this and other notebooks by simply importing the class and instantiating a model:

```
    from models import Net
    net = Net()
```

```
In [1]: # load the data if you need to; if you have already loaded the data, you may comment thi
        # -- DO NOT CHANGE THIS CELL -- #
        !mkdir /data
        !wget -P /data/ https://s3.amazonaws.com/video.udacity-data.com/topher/2018/May/5aea1b91
        !unzip -n /data/train-test-data.zip -d /data
```

```
mkdir: cannot create directory /data: File exists
--2020-05-16 19:30:03--  https://s3.amazonaws.com/video.udacity-data.com/topher/2018/May/5aea1b9
Resolving s3.amazonaws.com (s3.amazonaws.com)... 52.216.140.62
Connecting to s3.amazonaws.com (s3.amazonaws.com)|52.216.140.62|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 338613624 (323M) [application/zip]
Saving to: /data/train-test-data.zip.4

train-test-data.zip 100%[===================>] 322.93M  72.8MB/s    in 4.4s

2020-05-16 19:30:08 (74.1 MB/s) - /data/train-test-data.zip.4 saved [338613624/338613624]
```

```
Archive:  /data/train-test-data.zip
```

Note: Workspaces automatically close connections after 30 minutes of inactivity (including inactivity while training!). Use the code snippet below to keep your workspace alive during training. (The active_session context manager is imported below.)

```
from workspace_utils import active_session

with active_session():
    train_model(num_epochs)
```

```
In [2]: # import the usual resources
        import matplotlib.pyplot as plt
        import numpy as np

        # import utilities to keep workspaces alive during model training
        from workspace_utils import active_session

        # watch for any changes in model.py, if it changes, re-load it automatically
        %load_ext autoreload
        %autoreload 2
```

```
In [3]: ## TODO: Define the Net in models.py

        import torch
        import torch.nn as nn
        import torch.nn.functional as F

        ## TODO: Once you've define the network, you can instantiate it
        # one example conv layer has been provided for you
        from models import Net

        net = Net()
        print(net)
```

```
/home/workspace/models.py:46: UserWarning: nn.init.xavier_uniform is now deprecated in favor of
  I.xavier_uniform(self.fc1.weight.data)


Net(
  (conv1): Conv2d(1, 32, kernel_size=(4, 4), stride=(1, 1))
  (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (dropout1): Dropout(p=0.1)
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
  (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
```

```
  (dropout2): Dropout(p=0.2)
  (conv3): Conv2d(64, 128, kernel_size=(2, 2), stride=(1, 1))
  (bn3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pool3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (dropout3): Dropout(p=0.3)
  (conv4): Conv2d(128, 256, kernel_size=(1, 1), stride=(1, 1))
  (bn4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pool4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (dropout4): Dropout(p=0.4)
  (fc1): Linear(in_features=43264, out_features=1000, bias=True)
  (bn5): BatchNorm1d(1000, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (dropout5): Dropout(p=0.5)
  (fc2): Linear(in_features=1000, out_features=1000, bias=True)
  (bn6): BatchNorm1d(1000, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (dropout6): Dropout(p=0.6)
  (fc3): Linear(in_features=1000, out_features=136, bias=True)
)


/home/workspace/models.py:47: UserWarning: nn.init.xavier_uniform is now deprecated in favor of
  I.xavier_uniform(self.fc2.weight.data)
/home/workspace/models.py:48: UserWarning: nn.init.xavier_uniform is now deprecated in favor of
  I.xavier_uniform(self.fc3.weight.data)
```

0.4 Transform the dataset

To prepare for training, create a transformed dataset of images and keypoints.

0.4.1 TODO: Define a data transform

In PyTorch, a convolutional neural network expects a torch image of a consistent size as input. For efficient training, and so your model's loss does not blow up during training, it is also suggested that you normalize the input images and keypoints. The necessary transforms have been defined in `data_load.py` and you **do not** need to modify these; take a look at this file (you'll see the same transforms that were defined and applied in Notebook 1).

To define the data transform below, use a composition of: 1. Rescaling and/or cropping the data, such that you are left with a square image (the suggested size is 224x224px) 2. Normalizing the images and keypoints; turning each RGB image into a grayscale image with a color range of [0, 1] and transforming the given keypoints into a range of [-1, 1] 3. Turning these images and keypoints into Tensors

These transformations have been defined in `data_load.py`, but it's up to you to call them and create a `data_transform` below. **This transform will be applied to the training data and, later, the test data**. It will change how you go about displaying these images and keypoints, but these steps are essential for efficient training.

As a note, should you want to perform data augmentation (which is optional in this project), and randomly rotate or shift these images, a square image size will be useful; rotating a 224x224 image by 90 degrees will result in the same shape of output.

In [4]:
```
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms, utils

# the dataset we created in Notebook 1 is copied in the helper file `data_load.py`
from data_load import FacialKeypointsDataset
# the transforms we defined in Notebook 1 are in the helper file `data_load.py`
from data_load import Rescale, RandomCrop, Normalize, ToTensor


## TODO: define the data_transform using transforms.Compose([all tx's, . , .])
# order matters! i.e. rescaling should come before a smaller crop
data_transform = transforms.Compose([Rescale(225),
                                     RandomCrop(224),
                                     Normalize(),
                                     ToTensor()
                                     ])

# testing that you've defined a transform
assert(data_transform is not None), 'Define a data_transform'
```

In [5]:
```
# create the transformed dataset
transformed_dataset = FacialKeypointsDataset(csv_file='/data/training_frames_keypoints.c
                                             root_dir='/data/training/',
                                             transform=data_transform)


print('Number of images: ', len(transformed_dataset))

# iterate through the transformed dataset and print some stats about the first few sampl
for i in range(4):
    sample = transformed_dataset[i]
    print(i, sample['image'].size(), sample['keypoints'].size())
```

```
Number of images:  3462
0 torch.Size([1, 224, 224]) torch.Size([68, 2])
1 torch.Size([1, 224, 224]) torch.Size([68, 2])
2 torch.Size([1, 224, 224]) torch.Size([68, 2])
3 torch.Size([1, 224, 224]) torch.Size([68, 2])


/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
```

0.5 Batching and loading data

Next, having defined the transformed dataset, we can use PyTorch's DataLoader class to load the training data in batches of whatever size as well as to shuffle the data for training the model. You can read more about the parameters of the DataLoader, in this documentation.

Batch size Decide on a good batch size for training your model. Try both small and large batch sizes and note how the loss decreases as the model trains. Too large a batch size may cause your model to crash and/or run out of memory while training.

Note for Windows users: Please change the `num_workers` to 0 or you may face some issues with your DataLoader failing.

```
In [6]: # load training data in batches
        batch_size = 10

        train_loader = DataLoader(transformed_dataset,
                                  batch_size=batch_size,
                                  shuffle=True,
                                  num_workers=4)
```

0.6 Before training

Take a look at how this model performs before it trains. You should see that the keypoints it predicts start off in one spot and don't match the keypoints on a face at all! It's interesting to visualize this behavior so that you can compare it to the model after training and see how the model has improved.

Load in the test dataset The test dataset is one that this model has *not* seen before, meaning it has not trained with these images. We'll load in this test data and before and after training, see how your model performs on this set!

To visualize this test data, we have to go through some un-transformation steps to turn our images into python images from tensors and to turn our keypoints back into a recognizable range.

```
In [7]: # load in the test data, using the dataset class
        # AND apply the data_transform you defined above

        # create the test dataset
        test_dataset = FacialKeypointsDataset(csv_file='/data/test_frames_keypoints.csv',
                                              root_dir='/data/test/',
                                              transform=data_transform)
```

```
In [8]: # load test data in batches
        batch_size = 10

        test_loader = DataLoader(test_dataset,
                                 batch_size=batch_size,
                                 shuffle=True,
                                 num_workers=4)
```

0.7 Apply the model on a test sample

To test the model on a test sample of data, you have to follow these steps: 1. Extract the image and ground truth keypoints from a sample 2. Wrap the image in a Variable, so that the net can process it as input and track how it changes as the image moves through the network. 3. Make sure the

image is a FloatTensor, which the model expects. 4. Forward pass the image through the net to get the predicted, output keypoints.

This function test how the network performs on the first batch of test data. It returns the images, the transformed images, the predicted keypoints (produced by the model), and the ground truth keypoints.

```
In [9]: # test the model on a batch of test images

        def net_sample_output():

            # iterate through the test dataset
            for i, sample in enumerate(test_loader):

                # get sample data: images and ground truth keypoints
                images = sample['image']
                key_pts = sample['keypoints']

                # convert images to FloatTensors
                images = images.type(torch.FloatTensor)

                # forward pass to get net output
                output_pts = net(images)

                # reshape to batch_size x 68 x 2 pts
                output_pts = output_pts.view(output_pts.size()[0], 68, -1)

                # break after first image is tested
                if i == 0:
                    return images, output_pts, key_pts
```

Debugging tips If you get a size or dimension error here, make sure that your network outputs the expected number of keypoints! Or if you get a Tensor type error, look into changing the above code that casts the data into float types: `images = images.type(torch.FloatTensor)`.

```
In [10]: # call the above function
         # returns: test images, test predicted keypoints, test ground truth keypoints
         test_images, test_outputs, gt_pts = net_sample_output()

         # print out the dimensions of the data to see if they make sense
         print(test_images.data.size())
         print(test_outputs.data.size())
         print(gt_pts.size())
```

```
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
```

```
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()


torch.Size([10, 1, 224, 224])
torch.Size([10, 68, 2])
torch.Size([10, 68, 2])
```

0.8 Visualize the predicted keypoints

Once we've had the model produce some predicted output keypoints, we can visualize these points in a way that's similar to how we've displayed this data before, only this time, we have to "un-transform" the image/keypoint data to display it.

Note that I've defined a *new* function, `show_all_keypoints` that displays a grayscale image, its predicted keypoints and its ground truth keypoints (if provided).

```
In [11]: def show_all_keypoints(image, predicted_key_pts, gt_pts=None):
             """Show image with predicted keypoints"""
             # image is grayscale
             plt.imshow(image, cmap='gray')
             plt.scatter(predicted_key_pts[:, 0], predicted_key_pts[:, 1], s=20, marker='.', c='
             # plot ground truth points as green pts
             if gt_pts is not None:
                 plt.scatter(gt_pts[:, 0], gt_pts[:, 1], s=20, marker='.', c='g')
```

Un-transformation Next, you'll see a helper function. `visualize_output` that takes in a batch of images, predicted keypoints, and ground truth keypoints and displays a set of those images and their true/predicted keypoints.

This function's main role is to take batches of image and keypoint data (the input and output of your CNN), and transform them into numpy images and un-normalized keypoints (x, y) for normal display. The un-transformation process turns keypoints and images into numpy arrays from Tensors *and* it undoes the keypoint normalization done in the Normalize() transform; it's assumed that you applied these transformations when you loaded your test data.

```
In [12]: # visualize the output
         # by default this shows a batch of 10 images
         def visualize_output(test_images, test_outputs, gt_pts=None, batch_size=10):

             for i in range(batch_size):
                 plt.figure(figsize=(20,10))
                 ax = plt.subplot(1, batch_size, i+1)

                 # un-transform the image data
                 image = test_images[i].data   # get the image from it's Variable wrapper
                 image = image.numpy()   # convert to numpy array from a Tensor
```

```
            image = np.transpose(image, (1, 2, 0))   # transpose to go from torch to numpy

            # un-transform the predicted key_pts data
            predicted_key_pts = test_outputs[i].data
            predicted_key_pts = predicted_key_pts.numpy()
            # undo normalization of keypoints
            predicted_key_pts = predicted_key_pts*50.0+100

            # plot ground truth points for comparison, if they exist
            ground_truth_pts = None
            if gt_pts is not None:
                ground_truth_pts = gt_pts[i]
                ground_truth_pts = ground_truth_pts*50.0+100

            # call show_all_keypoints
            show_all_keypoints(np.squeeze(image), predicted_key_pts, ground_truth_pts)

            plt.axis('off')

        plt.show()

# call it
visualize_output(test_images, test_outputs, gt_pts)
```

0.9 Training

Loss function Training a network to predict keypoints is different than training a network to predict a class; instead of outputting a distribution of classes and using cross entropy loss, you may want to choose a loss function that is suited for regression, which directly compares a predicted value and target value. Read about the various kinds of loss functions (like MSE or L1/SmoothL1 loss) in this documentation.

0.9.1 TODO: Define the loss and optimization

Next, you'll define how the model will train by deciding on the loss function and optimizer.

```
In [13]: ## TODO: Define the loss and optimization
         import torch.optim as optim

         criterion = nn.SmoothL1Loss()

         optimizer = optim.Adam(net.parameters(), lr=0.001)
```

0.10 Training and Initial Observation

Now, you'll train on your batched training data from `train_loader` for a number of epochs.

To quickly observe how your model is training and decide on whether or not you should modify it's structure or hyperparameters, you're encouraged to start off with just one or two epochs at first. As you train, note how your the model's loss behaves over time: does it decrease quickly at first and then slow down? Does it take a while to decrease in the first place? What happens if you change the batch size of your training data or modify your loss function? etc.

Use these initial observations to make changes to your model and decide on the best architecture before you train for many epochs and create a final model.

In [14]:
```
def train_net(n_epochs):

    # prepare the net for training
    net.train()

    for epoch in range(n_epochs):  # loop over the dataset multiple times

        running_loss = 0.0

        # train on batches of data, assumes you already have train_loader
        for batch_i, data in enumerate(train_loader):
            # get the input images and their corresponding labels
            images = data['image']
            key_pts = data['keypoints']

            # flatten pts
            key_pts = key_pts.view(key_pts.size(0), -1)

            # convert variables to floats for regression loss
            key_pts = key_pts.type(torch.FloatTensor)
            images = images.type(torch.FloatTensor)

            # forward pass to get outputs
            output_pts = net(images)

            # calculate the loss between predicted and target keypoints
            loss = criterion(output_pts, key_pts)

            # zero the parameter (weight) gradients
            optimizer.zero_grad()

            # backward pass to calculate the weight gradients
            loss.backward()

            # update the weights
            optimizer.step()

            # print loss statistics
            running_loss += loss.item()
            if batch_i % 40 == 39:    # print every 10 batches
                print('Epoch: {}, Batch: {}, Avg. Loss: {}'.format(epoch + 1, batch_i+1
                running_loss = 0.0

    print('Finished Training')
```

In [15]:
```
# train your network
n_epochs = 30 # start small, and increase when you've decided on your model structure a
```

```
        # this is a Workspaces-specific context manager to keep the connection
        # alive while training your model, not part of pytorch
        with active_session():
            train_net(n_epochs)
```

```
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()


Epoch: 1, Batch: 40, Avg. Loss: 0.43838865160942075
Epoch: 1, Batch: 80, Avg. Loss: 0.2969559535384178
Epoch: 1, Batch: 120, Avg. Loss: 0.24619200117886067
Epoch: 1, Batch: 160, Avg. Loss: 0.21436508670449256
Epoch: 1, Batch: 200, Avg. Loss: 0.18505638055503368
Epoch: 1, Batch: 240, Avg. Loss: 0.15919912289828062
Epoch: 1, Batch: 280, Avg. Loss: 0.14504086710512637
Epoch: 1, Batch: 320, Avg. Loss: 0.13482975047081708


/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()


Epoch: 2, Batch: 40, Avg. Loss: 0.11853114720433951
Epoch: 2, Batch: 80, Avg. Loss: 0.10273864977061749
Epoch: 2, Batch: 120, Avg. Loss: 0.09765050075948238
Epoch: 2, Batch: 160, Avg. Loss: 0.09603990018367767
Epoch: 2, Batch: 200, Avg. Loss: 0.08062349827960133
Epoch: 2, Batch: 240, Avg. Loss: 0.07250503581017256
Epoch: 2, Batch: 280, Avg. Loss: 0.07170904967933893
Epoch: 2, Batch: 320, Avg. Loss: 0.07659287760034203


/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
```

```
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()


Epoch: 3, Batch: 40, Avg. Loss: 0.05955395177006721
Epoch: 3, Batch: 80, Avg. Loss: 0.056094863824546336
Epoch: 3, Batch: 120, Avg. Loss: 0.05639385986141861
Epoch: 3, Batch: 160, Avg. Loss: 0.0535178316757083
Epoch: 3, Batch: 200, Avg. Loss: 0.052713997475802896
Epoch: 3, Batch: 240, Avg. Loss: 0.045411446643993256
Epoch: 3, Batch: 280, Avg. Loss: 0.0480502896476537
Epoch: 3, Batch: 320, Avg. Loss: 0.047303524892777206


/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()


Epoch: 4, Batch: 40, Avg. Loss: 0.039159883325919506
Epoch: 4, Batch: 80, Avg. Loss: 0.03646521507762372
Epoch: 4, Batch: 120, Avg. Loss: 0.04054647982120514
Epoch: 4, Batch: 160, Avg. Loss: 0.04157778797671199
Epoch: 4, Batch: 200, Avg. Loss: 0.04166492493823171
Epoch: 4, Batch: 240, Avg. Loss: 0.04138366798870265
Epoch: 4, Batch: 280, Avg. Loss: 0.03991829375736415
Epoch: 4, Batch: 320, Avg. Loss: 0.03183107599616051


/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()


Epoch: 5, Batch: 40, Avg. Loss: 0.03986014956608415
Epoch: 5, Batch: 80, Avg. Loss: 0.03534009447321296
Epoch: 5, Batch: 120, Avg. Loss: 0.03282363857142627
```

```
Epoch: 5, Batch: 160, Avg. Loss: 0.03964651715941727
Epoch: 5, Batch: 200, Avg. Loss: 0.032495193369686606
Epoch: 5, Batch: 240, Avg. Loss: 0.03788599455729127
Epoch: 5, Batch: 280, Avg. Loss: 0.030737245036289097
Epoch: 5, Batch: 320, Avg. Loss: 0.031079653976485135


/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()


Epoch: 6, Batch: 40, Avg. Loss: 0.03251481968909502
Epoch: 6, Batch: 80, Avg. Loss: 0.0344493435928598
Epoch: 6, Batch: 120, Avg. Loss: 0.03041414057370275
Epoch: 6, Batch: 160, Avg. Loss: 0.031894429214298727
Epoch: 6, Batch: 200, Avg. Loss: 0.027755578514188528
Epoch: 6, Batch: 240, Avg. Loss: 0.0316761408932507
Epoch: 6, Batch: 280, Avg. Loss: 0.03250399243552238
Epoch: 6, Batch: 320, Avg. Loss: 0.03310151391196996


/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()


Epoch: 7, Batch: 40, Avg. Loss: 0.02473838026635349
Epoch: 7, Batch: 80, Avg. Loss: 0.026423647697083653
Epoch: 7, Batch: 120, Avg. Loss: 0.03473093782085925
Epoch: 7, Batch: 160, Avg. Loss: 0.03161564408801496
Epoch: 7, Batch: 200, Avg. Loss: 0.02865197635255754
Epoch: 7, Batch: 240, Avg. Loss: 0.036419388954527676
Epoch: 7, Batch: 280, Avg. Loss: 0.029996047681197523
Epoch: 7, Batch: 320, Avg. Loss: 0.025911838188767432


/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
```

```
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()


Epoch: 8, Batch: 40, Avg. Loss: 0.026432374003343286
Epoch: 8, Batch: 80, Avg. Loss: 0.025630361610092224
Epoch: 8, Batch: 120, Avg. Loss: 0.021440429193899036
Epoch: 8, Batch: 160, Avg. Loss: 0.029330301750451326
Epoch: 8, Batch: 200, Avg. Loss: 0.02893175391945988
Epoch: 8, Batch: 240, Avg. Loss: 0.028731807228177787
Epoch: 8, Batch: 280, Avg. Loss: 0.03451171151828021
Epoch: 8, Batch: 320, Avg. Loss: 0.02844139530789107


/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()


Epoch: 9, Batch: 40, Avg. Loss: 0.028847514605149626
Epoch: 9, Batch: 80, Avg. Loss: 0.028236552933230998
Epoch: 9, Batch: 120, Avg. Loss: 0.025470065465196968
Epoch: 9, Batch: 160, Avg. Loss: 0.028928882000036536
Epoch: 9, Batch: 200, Avg. Loss: 0.027559455554001033
Epoch: 9, Batch: 240, Avg. Loss: 0.03117499086074531
Epoch: 9, Batch: 280, Avg. Loss: 0.028395402524620295
Epoch: 9, Batch: 320, Avg. Loss: 0.024604409490711986


/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()


Epoch: 10, Batch: 40, Avg. Loss: 0.032297331001609565
Epoch: 10, Batch: 80, Avg. Loss: 0.0331166320014745
```

```
Epoch: 10, Batch: 120, Avg. Loss: 0.039064342994242905
Epoch: 10, Batch: 160, Avg. Loss: 0.036774706561118364
Epoch: 10, Batch: 200, Avg. Loss: 0.02871500593610108
Epoch: 10, Batch: 240, Avg. Loss: 0.02806642760988325
Epoch: 10, Batch: 280, Avg. Loss: 0.02689813682809472
Epoch: 10, Batch: 320, Avg. Loss: 0.028105723904445767


/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()


Epoch: 11, Batch: 40, Avg. Loss: 0.03137030410580337
Epoch: 11, Batch: 80, Avg. Loss: 0.026857529231347144
Epoch: 11, Batch: 120, Avg. Loss: 0.031576408469118175
Epoch: 11, Batch: 160, Avg. Loss: 0.02274198946543038
Epoch: 11, Batch: 200, Avg. Loss: 0.02093190746381879
Epoch: 11, Batch: 240, Avg. Loss: 0.027451572637073696
Epoch: 11, Batch: 280, Avg. Loss: 0.022440718533471227
Epoch: 11, Batch: 320, Avg. Loss: 0.03665755279362202


/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()


Epoch: 12, Batch: 40, Avg. Loss: 0.02224368171300739
Epoch: 12, Batch: 80, Avg. Loss: 0.02957735718227923
Epoch: 12, Batch: 120, Avg. Loss: 0.02954599114600569
Epoch: 12, Batch: 160, Avg. Loss: 0.024220777093432844
Epoch: 12, Batch: 200, Avg. Loss: 0.03120158815290779
Epoch: 12, Batch: 240, Avg. Loss: 0.028188830194994807
Epoch: 12, Batch: 280, Avg. Loss: 0.02712583604734391
Epoch: 12, Batch: 320, Avg. Loss: 0.02201195093803108


/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
```

```
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()


Epoch: 13, Batch: 40, Avg. Loss: 0.024309091409668325
Epoch: 13, Batch: 80, Avg. Loss: 0.024768831953406335
Epoch: 13, Batch: 120, Avg. Loss: 0.029148983559571205
Epoch: 13, Batch: 160, Avg. Loss: 0.029619447560980915
Epoch: 13, Batch: 200, Avg. Loss: 0.02577840054873377
Epoch: 13, Batch: 240, Avg. Loss: 0.023738685343414546
Epoch: 13, Batch: 280, Avg. Loss: 0.021739994129166006
Epoch: 13, Batch: 320, Avg. Loss: 0.02314617051742971


/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()


Epoch: 14, Batch: 40, Avg. Loss: 0.043872253084555266
Epoch: 14, Batch: 80, Avg. Loss: 0.044738140678964554
Epoch: 14, Batch: 120, Avg. Loss: 0.04064590479247272
Epoch: 14, Batch: 160, Avg. Loss: 0.03510775375179946
Epoch: 14, Batch: 200, Avg. Loss: 0.04039742087479681
Epoch: 14, Batch: 240, Avg. Loss: 0.033540586358867586
Epoch: 14, Batch: 280, Avg. Loss: 0.03378322096541524
Epoch: 14, Batch: 320, Avg. Loss: 0.035076766880229114


/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()


Epoch: 15, Batch: 40, Avg. Loss: 0.028838587831705807
Epoch: 15, Batch: 80, Avg. Loss: 0.030798912281170487
```

```
Epoch: 15, Batch: 120, Avg. Loss: 0.04046486255247146
Epoch: 15, Batch: 160, Avg. Loss: 0.03348453778307885
Epoch: 15, Batch: 200, Avg. Loss: 0.028063113312236966
Epoch: 15, Batch: 240, Avg. Loss: 0.035777456942014396
Epoch: 15, Batch: 280, Avg. Loss: 0.021576326875947415
Epoch: 15, Batch: 320, Avg. Loss: 0.029972789785824717


/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()


Epoch: 16, Batch: 40, Avg. Loss: 0.028513885871507227
Epoch: 16, Batch: 80, Avg. Loss: 0.02551206408534199
Epoch: 16, Batch: 120, Avg. Loss: 0.024632954015396537
Epoch: 16, Batch: 160, Avg. Loss: 0.025042111775837837
Epoch: 16, Batch: 200, Avg. Loss: 0.033349472819827496
Epoch: 16, Batch: 240, Avg. Loss: 0.028420043073128908
Epoch: 16, Batch: 280, Avg. Loss: 0.02609542680438608
Epoch: 16, Batch: 320, Avg. Loss: 0.027074257913045584


/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()


Epoch: 17, Batch: 40, Avg. Loss: 0.022764516971074046
Epoch: 17, Batch: 80, Avg. Loss: 0.02797442104201764
Epoch: 17, Batch: 120, Avg. Loss: 0.029651120700873435
Epoch: 17, Batch: 160, Avg. Loss: 0.024903904716484248
Epoch: 17, Batch: 200, Avg. Loss: 0.028904944681562482
Epoch: 17, Batch: 240, Avg. Loss: 0.023994145123288034
Epoch: 17, Batch: 280, Avg. Loss: 0.02478701921645552
Epoch: 17, Batch: 320, Avg. Loss: 0.023526296461932363


/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
```

```
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()


Epoch: 18, Batch: 40, Avg. Loss: 0.025363596132956447
Epoch: 18, Batch: 80, Avg. Loss: 0.025991727644577623
Epoch: 18, Batch: 120, Avg. Loss: 0.02514432524330914
Epoch: 18, Batch: 160, Avg. Loss: 0.0240170868113637
Epoch: 18, Batch: 200, Avg. Loss: 0.02624751809053123
Epoch: 18, Batch: 240, Avg. Loss: 0.023067806265316904
Epoch: 18, Batch: 280, Avg. Loss: 0.02275515552610159
Epoch: 18, Batch: 320, Avg. Loss: 0.022726708627305924


/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()


Epoch: 19, Batch: 40, Avg. Loss: 0.022100270504597574
Epoch: 19, Batch: 80, Avg. Loss: 0.02714331345632672
Epoch: 19, Batch: 120, Avg. Loss: 0.022714120545424522
Epoch: 19, Batch: 160, Avg. Loss: 0.023593331733718515
Epoch: 19, Batch: 200, Avg. Loss: 0.02270688789431006
Epoch: 19, Batch: 240, Avg. Loss: 0.02123074575792998
Epoch: 19, Batch: 280, Avg. Loss: 0.021482280734926463
Epoch: 19, Batch: 320, Avg. Loss: 0.025504079111851752


/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()


Epoch: 20, Batch: 40, Avg. Loss: 0.026208930811844767
Epoch: 20, Batch: 80, Avg. Loss: 0.026513059134595097
```

```
Epoch: 20, Batch: 120, Avg. Loss: 0.028017680789344013
Epoch: 20, Batch: 160, Avg. Loss: 0.027779324166476726
Epoch: 20, Batch: 200, Avg. Loss: 0.02725387883838266
Epoch: 20, Batch: 240, Avg. Loss: 0.02246393697569147
Epoch: 20, Batch: 280, Avg. Loss: 0.02757729860022664
Epoch: 20, Batch: 320, Avg. Loss: 0.02772145615890622


/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()


Epoch: 21, Batch: 40, Avg. Loss: 0.02081056012539193
Epoch: 21, Batch: 80, Avg. Loss: 0.028259566868655384
Epoch: 21, Batch: 120, Avg. Loss: 0.023437346960417927
Epoch: 21, Batch: 160, Avg. Loss: 0.0234342465759255
Epoch: 21, Batch: 200, Avg. Loss: 0.022779590729624034
Epoch: 21, Batch: 240, Avg. Loss: 0.019171385071240367
Epoch: 21, Batch: 280, Avg. Loss: 0.019919054326601328
Epoch: 21, Batch: 320, Avg. Loss: 0.02851435407064855


/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()


Epoch: 22, Batch: 40, Avg. Loss: 0.024463949375785887
Epoch: 22, Batch: 80, Avg. Loss: 0.02346388155128807
Epoch: 22, Batch: 120, Avg. Loss: 0.026886512502096595
Epoch: 22, Batch: 160, Avg. Loss: 0.020614680252037942
Epoch: 22, Batch: 200, Avg. Loss: 0.020317195216193795
Epoch: 22, Batch: 240, Avg. Loss: 0.024239321635104717
Epoch: 22, Batch: 280, Avg. Loss: 0.027951525663957
Epoch: 22, Batch: 320, Avg. Loss: 0.022375544381793587


/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
```

```
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()


Epoch: 23, Batch: 40, Avg. Loss: 0.023496167047414928
Epoch: 23, Batch: 80, Avg. Loss: 0.02096387220080942
Epoch: 23, Batch: 120, Avg. Loss: 0.02346103573217988
Epoch: 23, Batch: 160, Avg. Loss: 0.02524054767563939
Epoch: 23, Batch: 200, Avg. Loss: 0.02552384970476851
Epoch: 23, Batch: 240, Avg. Loss: 0.026374498941004278
Epoch: 23, Batch: 280, Avg. Loss: 0.02224802228156477
Epoch: 23, Batch: 320, Avg. Loss: 0.02657074499875307


/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()


Epoch: 24, Batch: 40, Avg. Loss: 0.024651240836828948
Epoch: 24, Batch: 80, Avg. Loss: 0.019230500585399567
Epoch: 24, Batch: 120, Avg. Loss: 0.03142862154636532
Epoch: 24, Batch: 160, Avg. Loss: 0.02257826739223674
Epoch: 24, Batch: 200, Avg. Loss: 0.020234108716249467
Epoch: 24, Batch: 240, Avg. Loss: 0.022268086555413902
Epoch: 24, Batch: 280, Avg. Loss: 0.021824287017807364
Epoch: 24, Batch: 320, Avg. Loss: 0.020400450006127356


/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()


Epoch: 25, Batch: 40, Avg. Loss: 0.02017104867845774
Epoch: 25, Batch: 80, Avg. Loss: 0.021253990847617386
```

```
Epoch: 25, Batch: 120, Avg. Loss: 0.02376696898136288
Epoch: 25, Batch: 160, Avg. Loss: 0.020607717987149953
Epoch: 25, Batch: 200, Avg. Loss: 0.03320752535946667
Epoch: 25, Batch: 240, Avg. Loss: 0.02065795655362308
Epoch: 25, Batch: 280, Avg. Loss: 0.018275745352730154
Epoch: 25, Batch: 320, Avg. Loss: 0.019987052772194147


/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()


Epoch: 26, Batch: 40, Avg. Loss: 0.018960943236015736
Epoch: 26, Batch: 80, Avg. Loss: 0.01882659865077585
Epoch: 26, Batch: 120, Avg. Loss: 0.023056212393566967
Epoch: 26, Batch: 160, Avg. Loss: 0.02326768762432039
Epoch: 26, Batch: 200, Avg. Loss: 0.018962465692311525
Epoch: 26, Batch: 240, Avg. Loss: 0.019716549047734587
Epoch: 26, Batch: 280, Avg. Loss: 0.023960360325872898
Epoch: 26, Batch: 320, Avg. Loss: 0.020341418869793414


/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()


Epoch: 27, Batch: 40, Avg. Loss: 0.02473912537097931
Epoch: 27, Batch: 80, Avg. Loss: 0.02184957512654364
Epoch: 27, Batch: 120, Avg. Loss: 0.019035840989090502
Epoch: 27, Batch: 160, Avg. Loss: 0.023854526423383503
Epoch: 27, Batch: 200, Avg. Loss: 0.019790423894301058
Epoch: 27, Batch: 240, Avg. Loss: 0.022766935895197092
Epoch: 27, Batch: 280, Avg. Loss: 0.023560947645455597
Epoch: 27, Batch: 320, Avg. Loss: 0.019236069498583676


/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
```

```
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()


Epoch: 28, Batch: 40, Avg. Loss: 0.02037574235582724
Epoch: 28, Batch: 80, Avg. Loss: 0.027043722663074733
Epoch: 28, Batch: 120, Avg. Loss: 0.022980872460175307
Epoch: 28, Batch: 160, Avg. Loss: 0.018664392456412316
Epoch: 28, Batch: 200, Avg. Loss: 0.01800965713337064
Epoch: 28, Batch: 240, Avg. Loss: 0.020653542736545204
Epoch: 28, Batch: 280, Avg. Loss: 0.020019731821957976
Epoch: 28, Batch: 320, Avg. Loss: 0.022304573527071624


/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()


Epoch: 29, Batch: 40, Avg. Loss: 0.022726566158235074
Epoch: 29, Batch: 80, Avg. Loss: 0.02134799809427932
Epoch: 29, Batch: 120, Avg. Loss: 0.01825169750954956
Epoch: 29, Batch: 160, Avg. Loss: 0.02568367514759302
Epoch: 29, Batch: 200, Avg. Loss: 0.023623251472599804
Epoch: 29, Batch: 240, Avg. Loss: 0.020490828121546657
Epoch: 29, Batch: 280, Avg. Loss: 0.01898943060077727
Epoch: 29, Batch: 320, Avg. Loss: 0.018838458345271647


/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()


Epoch: 30, Batch: 40, Avg. Loss: 0.021451014571357518
Epoch: 30, Batch: 80, Avg. Loss: 0.025475686776917427
```

```
Epoch: 30, Batch: 120, Avg. Loss: 0.018309946439694615
Epoch: 30, Batch: 160, Avg. Loss: 0.022613789164461197
Epoch: 30, Batch: 200, Avg. Loss: 0.018710011872462927
Epoch: 30, Batch: 240, Avg. Loss: 0.024006750143598765
Epoch: 30, Batch: 280, Avg. Loss: 0.018359832081478088
Epoch: 30, Batch: 320, Avg. Loss: 0.025064978934824466
Finished Training
```

0.11 Test data

See how your model performs on previously unseen, test data. We've already loaded and transformed this data, similar to the training data. Next, run your trained model on these images to see what kind of keypoints are produced. You should be able to see if your model is fitting each new face it sees, if the points are distributed randomly, or if the points have actually overfitted the training data and do not generalize.

In [16]:
```
# get a sample of test data again
test_images, test_outputs, gt_pts = net_sample_output()

print(test_images.data.size())
print(test_outputs.data.size())
print(gt_pts.size())
```

```
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()
/home/workspace/data_load.py:39: FutureWarning: Method .as_matrix will be removed in a future ve
  key_pts = self.key_pts_frame.iloc[idx, 1:].as_matrix()


torch.Size([10, 1, 224, 224])
torch.Size([10, 68, 2])
torch.Size([10, 68, 2])
```

In [17]:
```
## TODO: visualize your test output
# you can use the same function as before, by un-commenting the line below:

visualize_output(test_images, test_outputs, gt_pts)
```

Once you've found a good model (or two), save your model so you can load it and use it later!

Save your models but please **delete any checkpoints and saved models before you submit your project** otherwise your workspace may be too large to submit.

In [18]:
```
## TODO: change the name to something uniqe for each new model
model_dir = 'saved_models/'
model_name = 'deep_keypoints_model.pt'

# after training, save your model parameters in the dir 'saved_models'
torch.save(net.state_dict(), model_dir+model_name)
```

After you've trained a well-performing model, answer the following questions so that we have some insight into your training and architecture selection process. Answering all questions is required to pass this project.

0.11.1 Question 1: What optimization and loss functions did you choose and why?

Answer: I used Adams optimizer as Adams has better performance than SGD. Used Smooth L1 Loss as it is less sensitive to outliers and thus won't overfit the dataset

0.11.2 Question 2: What kind of network architecture did you start with and how did it change as you tried different architectures? Did you decide to add more convolutional layers or any layers to avoid overfitting the data?

Answer: Started with a basic conv-maxpool-dense layer architecture and trained the model. The initial architecture was itself pretty accurate. Thus, made the architecture deeper and added more conv-maxpool-dense layers

0.11.3 Question 3: How did you decide on the number of epochs and batch_size to train your model?

Answer: Initially, I ran everything on 1 epoch while observing the loss. Found out the ideal architecture and then scaled the number of epochs. In terms of batch size, I proceeded with the standard 10 value as it would take up a lot of time if we increase the batch size.

0.12 Feature Visualization

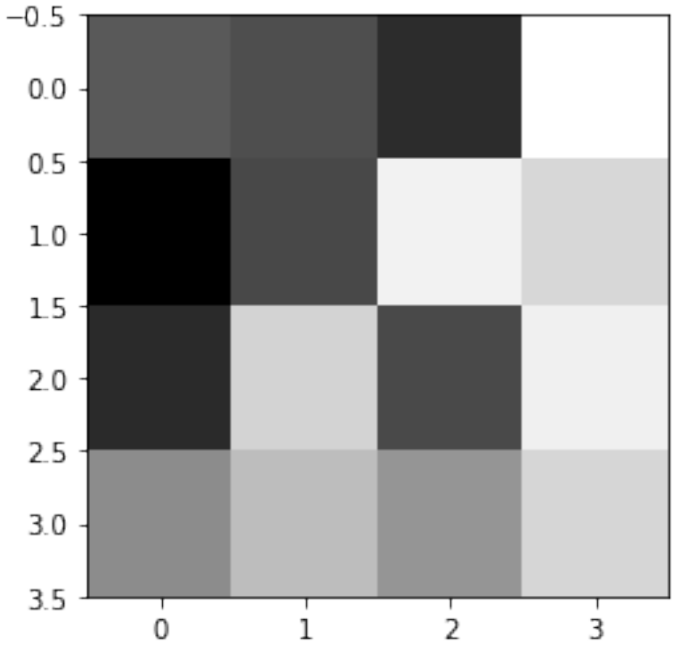
Sometimes, neural networks are thought of as a black box, given some input, they learn to produce some output. CNN's are actually learning to recognize a variety of spatial patterns and you can visualize what each convolutional layer has been trained to recognize by looking at the weights that make up each convolutional kernel and applying those one at a time to a sample image. This technique is called feature visualization and it's useful for understanding the inner workings of a CNN.

In the cell below, you can see how to extract a single filter (by index) from your first convolutional layer. The filter should appear as a grayscale grid.

```
In [19]: # Get the weights in the first conv layer, "conv1"
         # if necessary, change this to reflect the name of your first conv layer
         weights1 = net.conv1.weight.data

         w = weights1.numpy()

         filter_index = 0

         print(w[filter_index][0])
         print(w[filter_index][0].shape)

         # display the filter weights
         plt.imshow(w[filter_index][0], cmap='gray')
```

```
[[-0.11348986 -0.13728651 -0.20896092  0.24026456]
 [-0.30323762 -0.14966816  0.21190016  0.15328155]
 [-0.21201837  0.14706743 -0.14577763  0.20770691]
 [-0.00460567  0.0999981   0.01409672  0.1512282 ]]
(4, 4)
```

Out[19]: <matplotlib.image.AxesImage at 0x7f9c02a727b8>



0.13 Feature maps

Each CNN has at least one convolutional layer that is composed of stacked filters (also known as convolutional kernels). As a CNN trains, it learns what weights to include in it's convolutional kernels and when these kernels are applied to some input image, they produce a set of **feature maps**. So, feature maps are just sets of filtered images; they are the images produced by applying a convolutional kernel to an input image. These maps show us the features that the different layers of the neural network learn to extract. For example, you might imagine a convolutional kernel that detects the vertical edges of a face or another one that detects the corners of eyes. You can see what kind of features each of these kernels detects by applying them to an image. One such example is shown below; from the way it brings out the lines in an the image, you might characterize this as an edge detection filter.

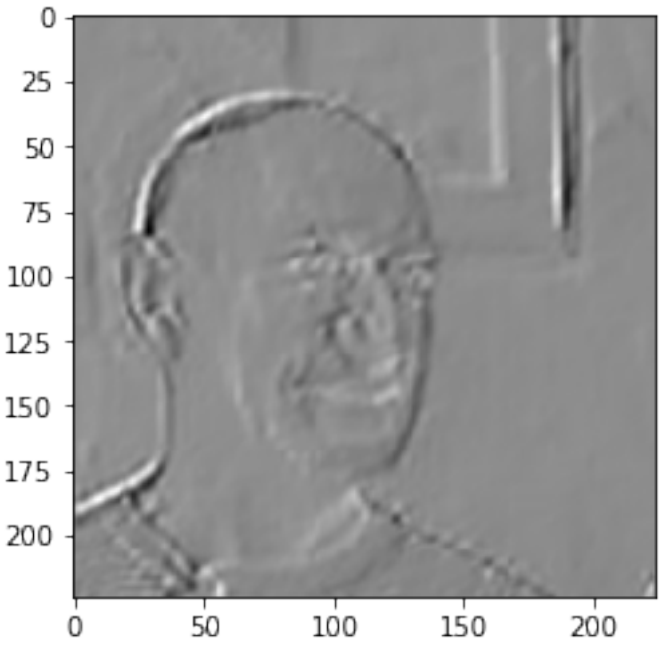
Next, choose a test image and filter it with one of the convolutional kernels in your trained CNN; look at the filtered output to get an idea what that particular kernel detects.

0.14 ### TODO: Filter an image to see the effect of a convolutional kernel

In [20]:
```
import cv2
##TODO: load in and display any image from the transformed test dataset

## TODO: Using cv's filter2D function,
## apply a specific set of filter weights (like the one displayed above) to the test im

weights1 = net.conv1.weight.data

w = weights1.numpy()

filter_index = 0

print(w[filter_index][0])
print(w[filter_index][0].shape)

# display the filter weights
# plt.imshow(w[filter_index][0], cmap='gray')


idx_img = 0
img = np.squeeze(test_images[idx_img].data.numpy())

# plt.imshow(img, cmap="gray")

filtered_img = cv2.filter2D(img, -1, w[filter_index][0])
plt.imshow(filtered_img, cmap="gray")
```

```
[[-0.11348986 -0.13728651 -0.20896092  0.24026456]
 [-0.30323762 -0.14966816  0.21190016  0.15328155]
 [-0.21201837  0.14706743 -0.14577763  0.20770691]
 [-0.00460567  0.0999981   0.01409672  0.1512282 ]]
(4, 4)
```

Out[20]: <matplotlib.image.AxesImage at 0x7f9c02aabbe0>



0.14.1 Question 4: Choose one filter from your trained CNN and apply it to a test image; what purpose do you think it plays? What kind of feature do you think it detects?

Answer: Looks like it blurs out noise

0.15 Moving on!

Now that you've defined and trained your model (and saved the best model), you are ready to move on to the last notebook, which combines a face detector with your saved model to create a facial keypoint detection system that can predict the keypoints on *any* face in an image!

In [ ]: