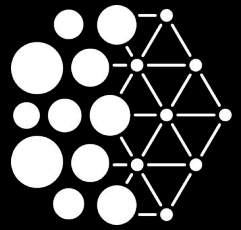
Institut
québécois
d'intelligence
artificielle



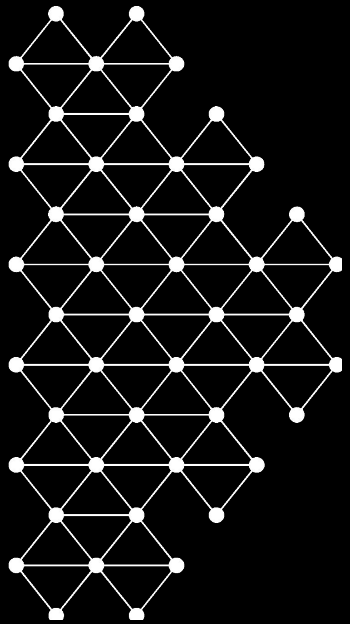
Mila

IFT 6759

Block 1 Common Presentations

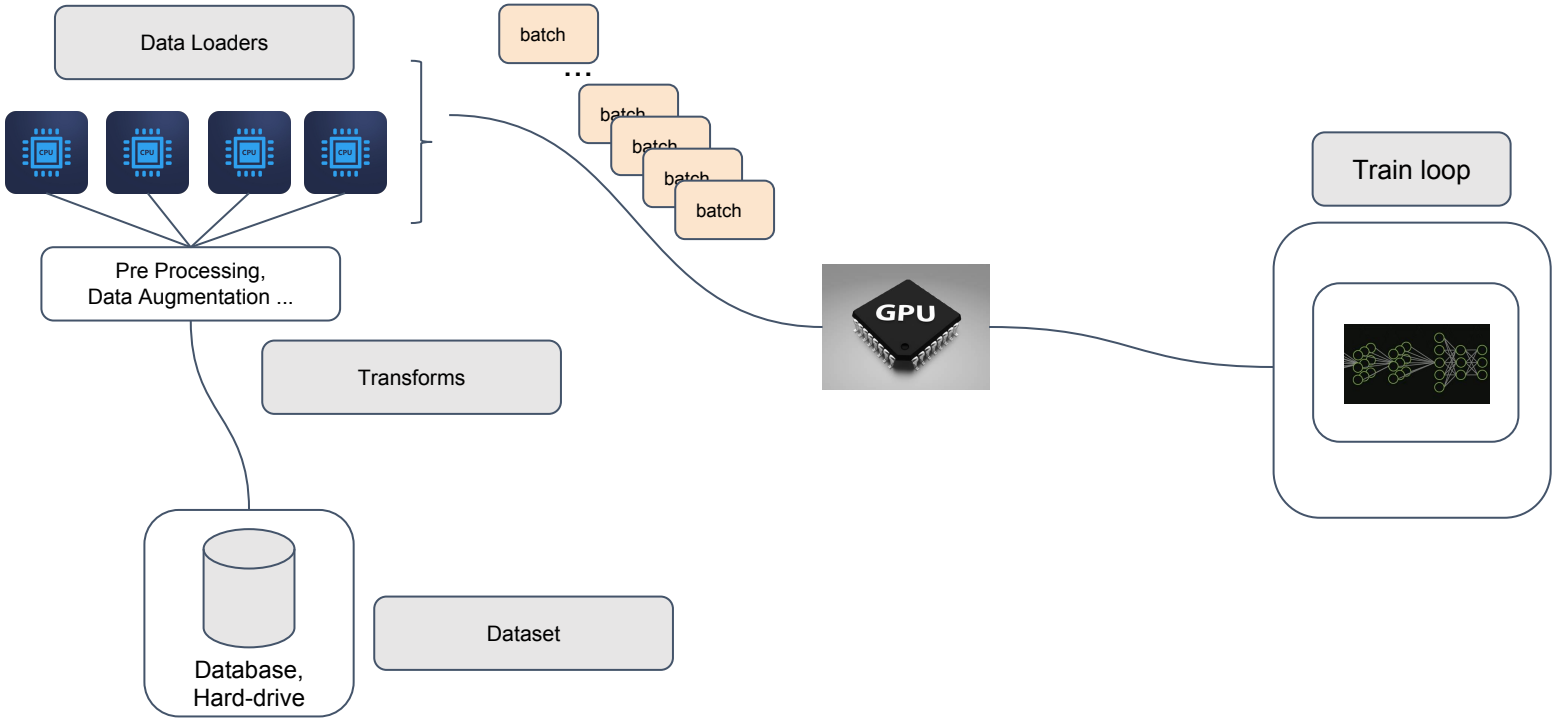
Jeremy Pinto

Institut
québécois d'intelligence
artificielle



Data Loaders and Data Visualization

Data Loaders



Mila

Datasets

- Typically expects a list or dataframe with filenames and labels in __init__()
- __getitem__() returns the loaded image and label
- Data augmentation + preprocessing can be specified as transforms in __getitem__()

```python
class FaceLandmarksDataset(Dataset):
    """Face Landmarks dataset."""

    def __init__(self, csv_file, root_dir, transform=None):
        """
        Args:
            csv_file (string): Path to the csv file with annotations.
            root_dir (string): Directory with all the images.
            transform (callable, optional): Optional transform to be applied
                on a sample.
        """
        self.landmarks_frame = pd.read_csv(csv_file)
        self.root_dir = root_dir
        self.transform = transform

    def __len__(self):
        return len(self.landmarks_frame)

    def __getitem__(self, idx):
        img_name = os.path.join(self.root_dir,
                                self.landmarks_frame.iloc[idx, 0])
        image = io.imread(img_name)
        landmarks = self.landmarks_frame.iloc[idx, 1:].as_matrix()
        landmarks = landmarks.astype('float').reshape(-1, 2)
        sample = {'image': image, 'landmarks': landmarks}

        if self.transform:
            sample = self.transform(sample)

        return sample
```

Data Augmentation

- Data augmentation + preprocessing can be specified here:

```
if self.transform:
    sample = self.transform(sample)

return sample
```

- Referred to as “transforms”
- Examples: cropping, rescaling, mean subtraction, etc.
- Transforms can be chained sequentially:

```
transforms.Compose([
    transforms.CenterCrop(10),
    transforms.ToTensor(),
])
```

```
class RandomCrop(object):
    """Crop randomly the image in a sample.

    Args:
        output_size (tuple or int): Desired output size. If int, square crop
            is made.
    """

    def __init__(self, output_size):
        assert isinstance(output_size, (int, tuple))
        if isinstance(output_size, int):
            self.output_size = (output_size, output_size)
        else:
            assert len(output_size) == 2
            self.output_size = output_size

    def __call__(self, sample):
        image, landmarks = sample['image'], sample['landmarks']

        h, w = image.shape[:2]
        new_h, new_w = self.output_size

        top = np.random.randint(0, h - new_h)
        left = np.random.randint(0, w - new_w)

        image = image[top: top + new_h,
                      left: left + new_w]

        landmarks = landmarks - [left, top]

        return {'image': image, 'landmarks': landmarks}
```

Data Loaders

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

PROS

- Generate batches on the fly from the previously defined dataset
- Can be used to specify split strategies (shuffle the data at each epoch, size of train vs. valid data, etc.)
- Easily define batch sizes, pythonic
- Asynchronous
- Low RAM usage

CONS

- Loading of data + processing needs to be faster than rate of GPU inference

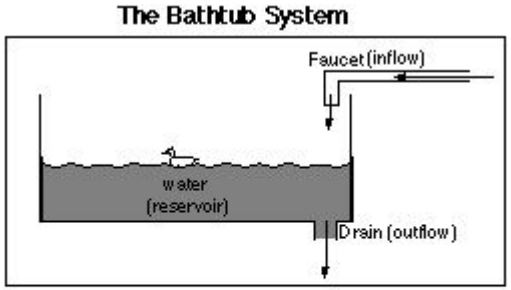


Figure 2.2. The Bathtub system is an open system with three components, an inflow, an outflow, and a reservoir; it is an open system since we do not keep track of the ultimate source and sink for water — we are only concerned with the water withinthe confines of the system portrayed in this drawing.

Mila

Big Picture

```
# Train the model
total_step = len(train_loader)
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        images = images.to(device)
        labels = labels.to(device)

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

The Zen of Python

References

- https://pytorch.org/tutorials/beginner/data_loading_tutorial.html
- https://stanford.edu/~shervine/blog/pytorch-how-to-generate-data-parallel
- https://pytorch.org/docs/master/data.html#torch.utils.data.Dataset
- https://pytorch.org/docs/master/data.html#torch.utils.data.DataLoader
- https://pytorch.org/docs/master/torchvision/datasets.html

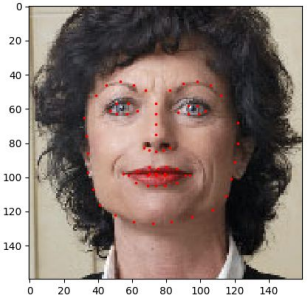
Data Visualization

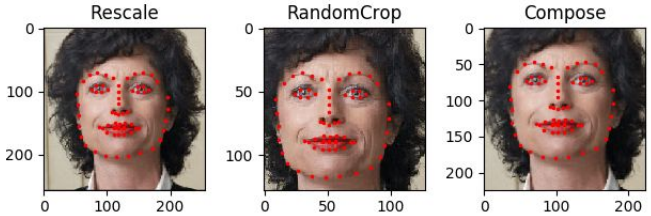
- Important for sanity checks
- Metrics don't always tell the whole / right story
- Provides insight in overfitting, dataset debugging
- Things to visualize:
  - Data quality (variance, noise, etc.)
  - Semantic meaning preservation when applying data transformations and augmentation
  - Class distributions / balanced dataset

plotly matplotlib

Visdom seaborn

Visualisation



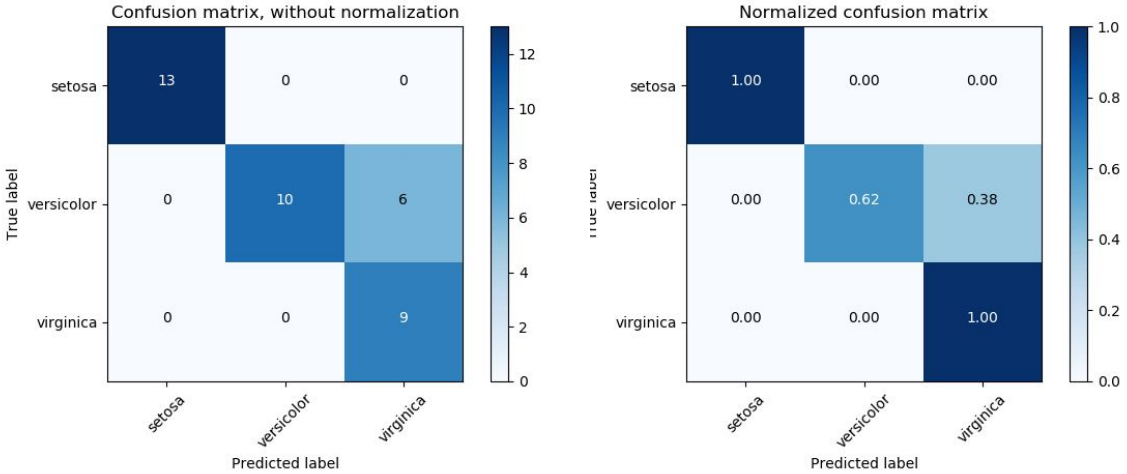


Mila

Confusion Matrix

- Useful to visualize errors in multi-class datasets
- Understand which classes get confused, which classes are good, can help visualize data imbalance

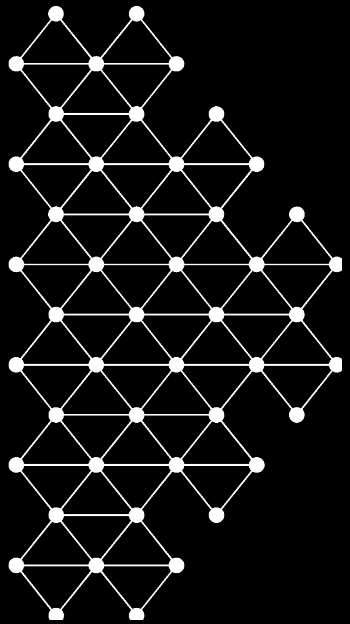
`sklearn.metrics.` **`confusion_matrix`** (*y_true, y_pred, labels=None, sample_weight=None*)



Mila

Institut
québécois d'intelligence
artificielle

Mila



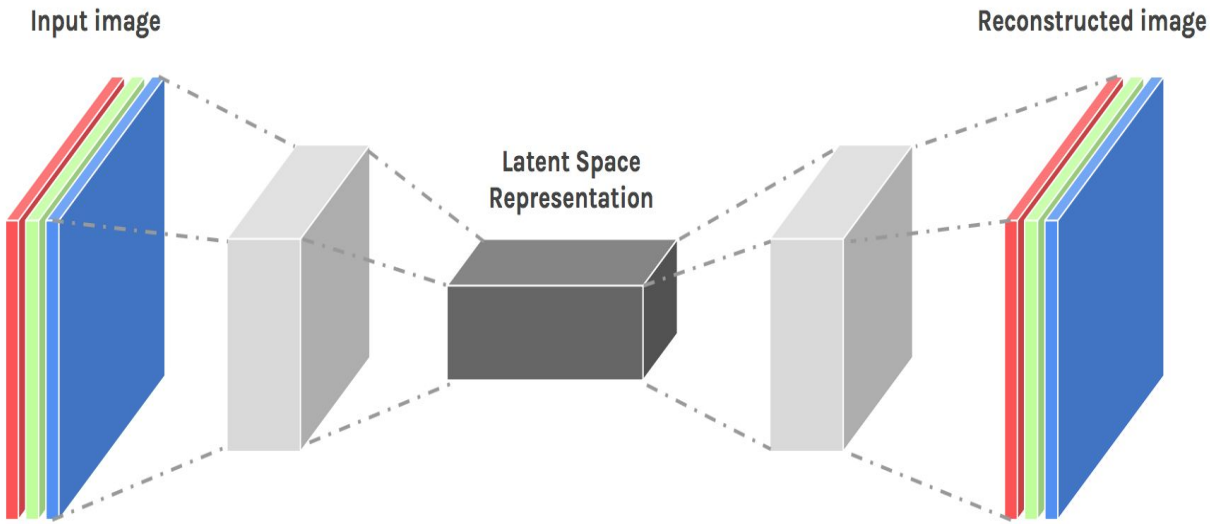
Dimensionality Reduction

Dimensionality Reduction

- Reduce data from high dimensionality (i.e. image) to a low-dimensional representation (i.e. vector)
- Keep only the “relevant” information
- Can lead to better results in classification and regression tasks
- Helps interpretability (for humans and machines!)
- Can be interpreted visually, good for sanity checks

Dimensionality Reduction

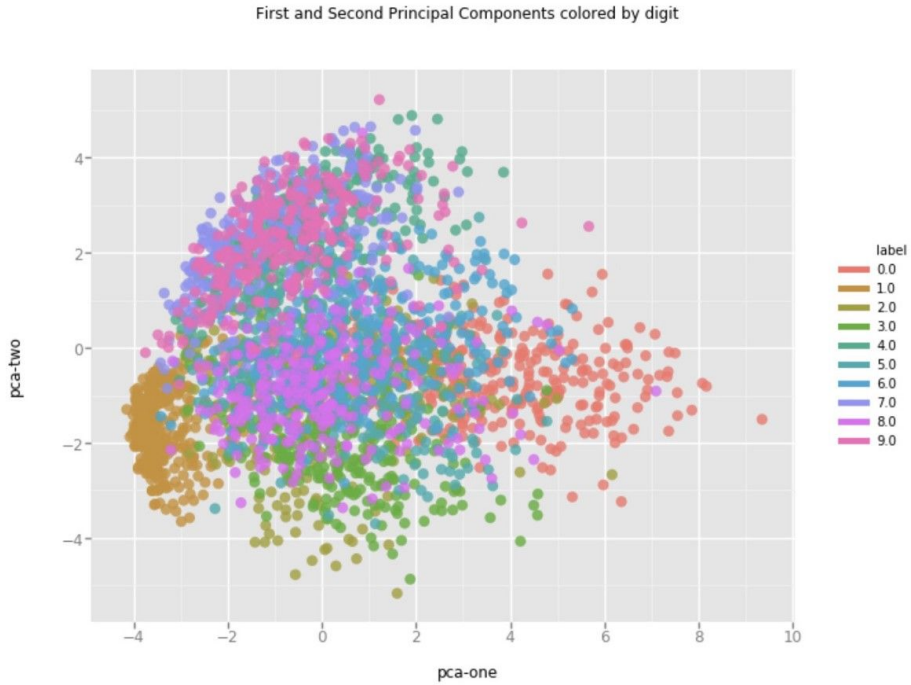
Example: reduce an image (i.e. 3x28x28=2352) to a vector representation (1xN) with N << 2352



Mila

Principal Component Analysis (PCA)

- Linear mapping of the data that maximizes the variance in the latent space
- The more “components” we use, the more information we keep (usually keep 30-50 components)
- First 2 components can be plotted visually
- Can use classes as “extra dimensions”
- Available in scikit-learn



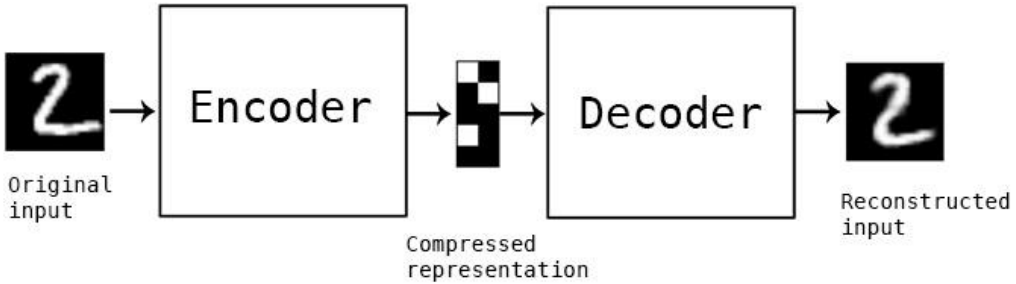
Sources https://en.wikipedia.org/wiki/Dimensionality_reduction,

https://medium.com/@luckylwk/visualising-high-dimensional-datasets-using-pca-and-t-sne-in-python-8ef87e7915b

Mila

Autoencoders

- Use neural networks to encode and decode the data
- The encoded layer is considered an intermediate, compressed format of the data
- NNs can use non-linear transformations
- Can be set up as an self-supervised problem

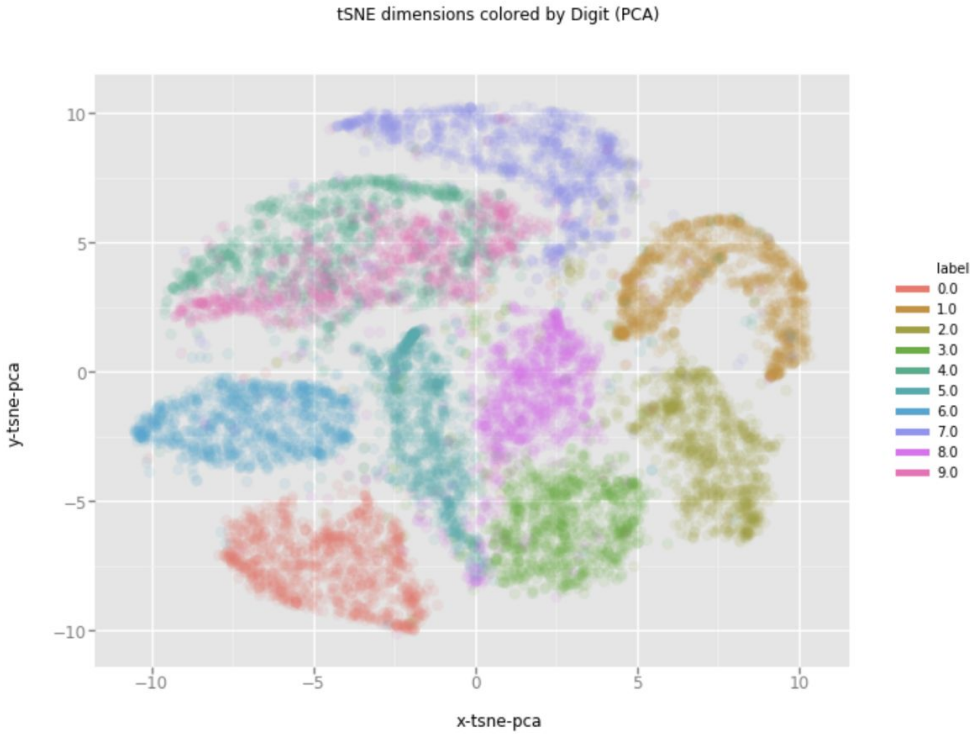


Sources https://blog.keras.io/building-autoencoders-in-keras.html

Mila

t-SNE

- Pronounced “tee-snee”
- Good algorithm for visualizing clusters based on data similarity
- Works best on low-dimensional data: start with PCA, autoencoders, etc.; then use t-SNE



Mila