zalandoresearch / **fashion-mnist**

A MNIST-like fashion product database. Benchmark 👉 http://fashion-mnist.s3-website.eu-ce...

#mnist #deep-learning #benchmark #machine-learning #dataset #computer-vision #fashion #fashion-mnist #gan #zalando #convolutional-neural-networks

201 commits | **1** branch | **0** releases | **4** contributors | MIT

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

hanxiao Update README.ja.md — Latest commit `629abc0` on Oct 5, 2018

benchmark	first commit	a year ago
data	fixes #44 I removed duplicate samples from the data set	a year ago
doc/img	update image	a year ago
static	add highlight to column header	a year ago
utils	simplified reader	a year ago
visualization	first commit	a year ago
.catwatch.yaml	add zappr and catwatch	a year ago
.dockerignore	first commit	a year ago
.gitignore	update readme	a year ago
.zappr.yaml	add zappr and catwatch	a year ago
CONTRIBUTING.md	first commit	a year ago
Dockerfile	first commit	a year ago
LICENSE	first commit	a year ago
MAINTAINERS	Update MAINTAINERS	10 months ago
README.ja.md	Update README.ja.md	3 months ago
README.md	Update README.md	3 months ago
README.zh-CN.md	Update README.zh-CN.md	3 months ago
app.py	first commit	a year ago
configs.py	first commit	a year ago
requirements.txt	first commit	a year ago

README.md

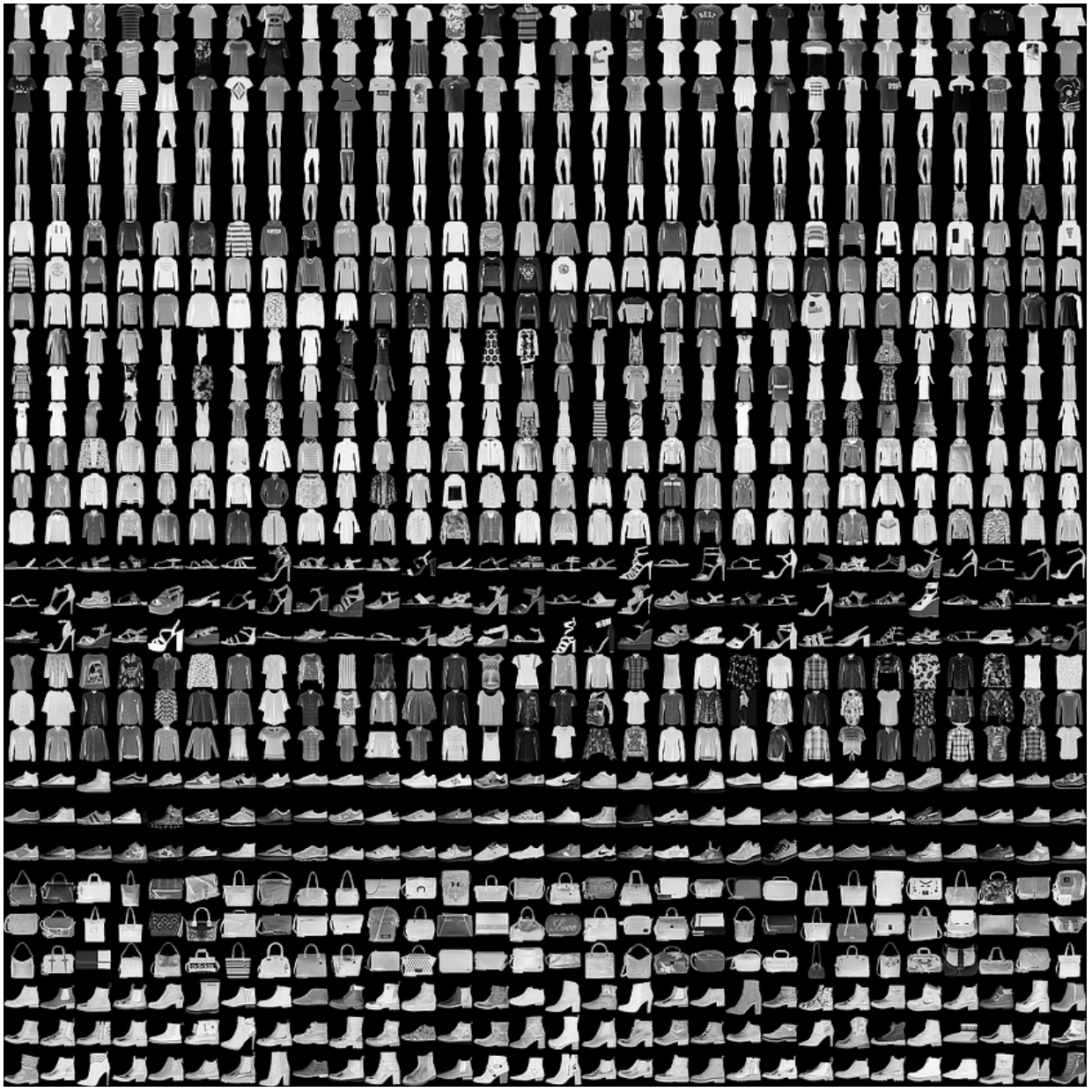
Fashion-MNIST

Star 5k | chat on gitter | README 中文 | README 日本語 | License MIT | Year in Review

▶ Table of Contents

`Fashion-MNIST` is a dataset of Zalando's article images—consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes. We intend `Fashion-MNIST` to serve as a direct **drop-in replacement** for the original MNIST dataset for benchmarking machine learning algorithms. It shares the same image size and structure of training and testing splits.

Here's an example how the data looks (*each class takes three-rows*):



Why we made Fashion-MNIST

The original MNIST dataset contains a lot of handwritten digits. Members of the AI/ML/Data Science community love this dataset and use it as a benchmark to validate their algorithms. In fact, MNIST is often the first dataset researchers try. "*If it doesn't work on MNIST, it **won't work** at all*", they said. "*Well, if it does work on MNIST, it may still fail on others.*"

To Serious Machine Learning Researchers

Seriously, we are talking about replacing MNIST. Here are some good reasons:

- **MNIST is too easy.** Convolutional nets can achieve 99.7% on MNIST. Classic machine learning algorithms can also achieve 97% easily. Check out our side-by-side benchmark for Fashion-MNIST vs. MNIST, and read "Most pairs of MNIST digits can be distinguished pretty well by just one pixel."
- **MNIST is overused.** In this April 2017 Twitter thread, Google Brain research scientist and deep learning expert Ian Goodfellow calls for people to move away from MNIST.
- **MNIST can not represent modern CV tasks**, as noted in this April 2017 Twitter thread, deep learning expert/Keras author François Chollet.

Get the Data

Many ML libraries already include Fashion-MNIST data/API, give it a try!

You can use direct links to download the dataset. The data is stored in the **same** format as the original MNIST data.

Name	Content	Examples	Size	Link	MD5 Checksum
`train-images-idx3-ubyte.gz`	training set images	60,000	26 MBytes	Download	`8d4fb7e6c68d591d4c3dfef9ec88bf0d`

Name	Content	Examples	Size	Link	MD5 Checksum
`train-labels-idx1-ubyte.gz`	training set labels	60,000	29 KBytes	Download	`25c81989df183df01b3e8a0aad5dffbe`
`t10k-images-idx3-ubyte.gz`	test set images	10,000	4.3 MBytes	Download	`bef4ecab320f06d8554ea6380940ec79`
`t10k-labels-idx1-ubyte.gz`	test set labels	10,000	5.1 KBytes	Download	`bb300cfdad3c16e7a12a480ee83cd310`

Alternatively, you can clone this GitHub repository; the dataset appears under `data/fashion`. This repo also contains some scripts for benchmark and visualization.

```
git clone git@github.com:zalandoresearch/fashion-mnist.git
```

Labels

Each training and test example is assigned to one of the following labels:

Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

Usage

Loading data with Python (requires NumPy)

Use `utils/mnist_reader` in this repo:

```
import mnist_reader
X_train, y_train = mnist_reader.load_mnist('data/fashion', kind='train')
X_test, y_test = mnist_reader.load_mnist('data/fashion', kind='t10k')
```

Loading data with Tensorflow

Make sure you have downloaded the data and placed it in `data/fashion`. Otherwise, *Tensorflow will download and use the original MNIST.*

```
from tensorflow.examples.tutorials.mnist import input_data
data = input_data.read_data_sets('data/fashion')
```

```
data.train.next_batch(BATCH_SIZE)
```

Note, Tensorflow supports passing in a source url to the `read_data_sets`. You may use:

```
data = input_data.read_data_sets('data/fashion', source_url='http://fashion-mnist.s3-website.eu-central-1.amazonaws.co
```

Also, an official Tensorflow tutorial of using `tf.keras`, a high-level API to train Fashion-MNIST can be found here.

Loading data with other machine learning libraries

To date, the following libraries have included `Fashion-MNIST` as a built-in dataset. Therefore, you don't need to download `Fashion-MNIST` by yourself. Just follow their API and you are ready to go.

- Apache MXNet Gluon
- deeplearn.js
- Kaggle
- Pytorch
- Keras
- Edward
- Tensorflow
- Torch
- JuliaML
- Chainer

You are welcome to make pull requests to other open-source machine learning packages, improving their support to `Fashion-MNIST` dataset.

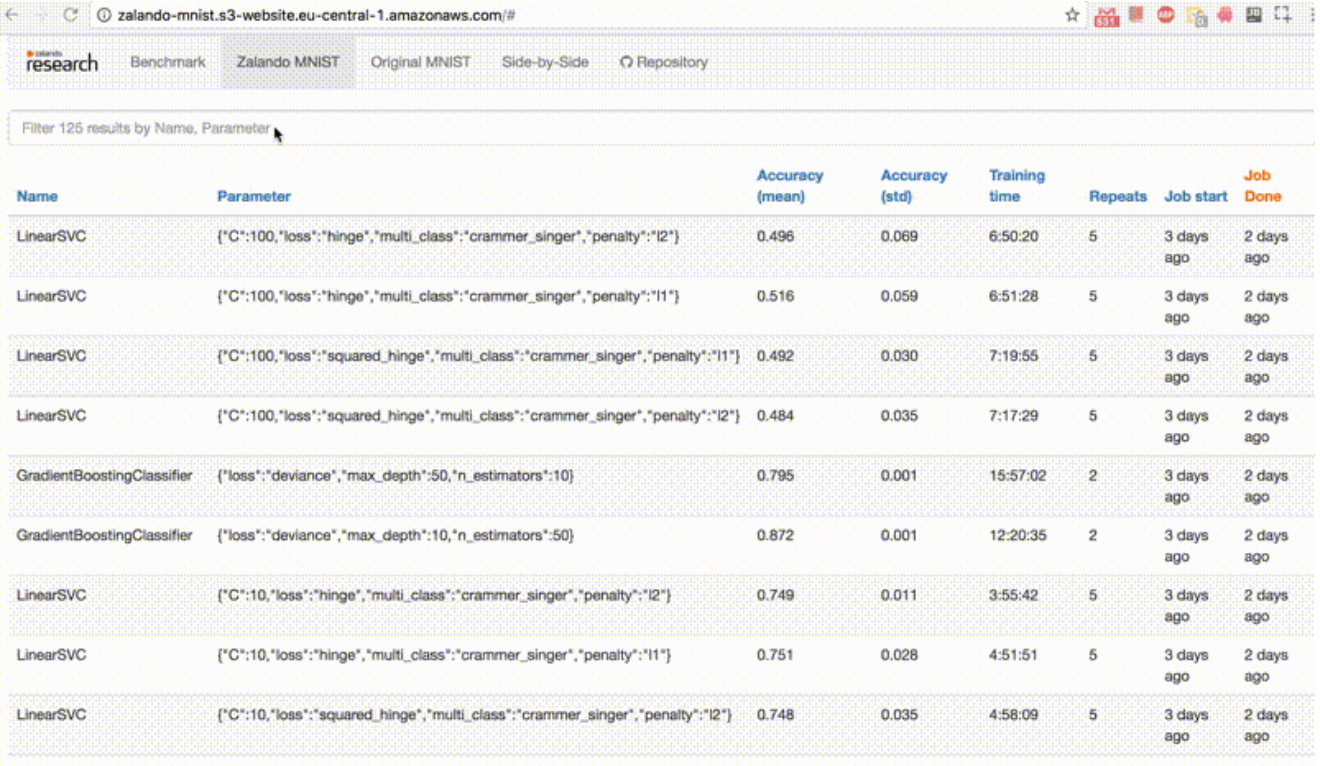
Loading data with other languages

As one of the Machine Learning community's most popular datasets, MNIST has inspired people to implement loaders in many different languages. You can use these loaders with the `Fashion-MNIST` dataset as well. (Note: may require decompressing first.) To date, we haven't yet tested all of these loaders with Fashion-MNIST.

- C
- C++
- Java
- Python and this and this
- Scala
- Go
- C#
- NodeJS and this
- Swift
- R and this
- Matlab
- Ruby

Benchmark

We built an automatic benchmarking system based on `scikit-learn` that covers 129 classifiers (but no deep learning) with different parameters. Find the results here.



Name	Parameter	Accuracy (mean)	Accuracy (std)	Training time	Repeats	Job start	Job Done
LinearSVC	{"C":100,"loss":"hinge","multi_class":"crammer_singer","penalty":"l2"}	0.496	0.069	6:50:20	5	3 days ago	2 days ago
LinearSVC	{"C":100,"loss":"hinge","multi_class":"crammer_singer","penalty":"l1"}	0.516	0.059	6:51:28	5	3 days ago	2 days ago
LinearSVC	{"C":100,"loss":"squared_hinge","multi_class":"crammer_singer","penalty":"l1"}	0.492	0.030	7:19:55	5	3 days ago	2 days ago
LinearSVC	{"C":100,"loss":"squared_hinge","multi_class":"crammer_singer","penalty":"l2"}	0.484	0.035	7:17:29	5	3 days ago	2 days ago
GradientBoostingClassifier	{"loss":"deviance","max_depth":50,"n_estimators":10}	0.795	0.001	15:57:02	2	3 days ago	2 days ago
GradientBoostingClassifier	{"loss":"deviance","max_depth":10,"n_estimators":50}	0.872	0.001	12:20:35	2	3 days ago	2 days ago
LinearSVC	{"C":10,"loss":"hinge","multi_class":"crammer_singer","penalty":"l2"}	0.749	0.011	3:55:42	5	3 days ago	2 days ago
LinearSVC	{"C":10,"loss":"hinge","multi_class":"crammer_singer","penalty":"l1"}	0.751	0.028	4:51:51	5	3 days ago	2 days ago
LinearSVC	{"C":10,"loss":"squared_hinge","multi_class":"crammer_singer","penalty":"l2"}	0.748	0.035	4:58:09	5	3 days ago	2 days ago

You can reproduce the results by running `benchmark/runner.py`. We recommend building and deploying this Dockerfile.

You are welcome to submit your benchmark; simply create a new issue and we'll list your results here. Before doing that, please make sure it does not already appear in this list. Visit our contributor guidelines for additional details.

The table below collects the submitted benchmarks. Note that **we haven't yet tested these results**. You are welcome to validate the results using the code provided by the submitter. Test accuracy may differ due to the number of epoch, batch size, etc. To correct this table, please create a new issue.

Classifier	Preprocessing	Fashion test accuracy	MNIST test accuracy	Submitter	Code
2 Conv+pooling	None	0.876	-	Kashif Rasul	🔗
2 Conv+pooling	None	0.916	-	Tensorflow's doc	🔗
2 Conv+pooling+ELU activation (PyTorch)	None	0.903	-	@AbhirajHinge	🔗
2 Conv	Normalization, random horizontal flip, random vertical flip, random translation, random rotation.	0.919	0.971	Kyriakos Efthymiadis	🔗
2 Conv <100K parameters	None	0.925	0.992	@hardmaru	🔗
2 Conv ~113K parameters	Normalization	0.922	0.993	Abel G.	🔗
2 Conv+3 FC ~1.8M parameters	Normalization	0.932	0.994	@Xfan1025	🔗
2 Conv+3 FC ~500K parameters	Augmentation, batch normalization	0.934	0.994	@cmasch	🔗

Classifier	Preprocessing	Fashion test accuracy	MNIST test accuracy	Submitter	Code
2 Conv+pooling+BN	None	0.934	-	@khanguyen1207	🔗
2 Conv+2 FC	Random Horizontal Flips	0.939	-	@ashmeet13	🔗
3 Conv+2 FC	None	0.907	-	@Cenk Bircanoğlu	🔗
3 Conv+pooling+BN	None	0.903	0.994	@meghanabhange	🔗
3 Conv+pooling+2 FC+dropout	None	0.926	-	@Umberto Griffo	🔗
3 Conv+BN+pooling	None	0.921	0.992	@GunjanChhablani	🔗
5 Conv+BN+pooling	None	0.931	-	@Noumanmufc1	🔗
CNN with optional shortcuts, dense-like connectivity	standardization+augmentation+random erasing	0.947	-	@kennivich	🔗
GRU+SVM	None	0.888	0.965	@AFAgarap	🔗
GRU+SVM with dropout	None	0.897	0.988	@AFAgarap	🔗
WRN40-4 8.9M params	standard preprocessing (mean/std subtraction/division) and augmentation (random crops/horizontal flips)	0.967	-	@ajbrock	🔗 🔗
DenseNet-BC 768K params	standard preprocessing (mean/std subtraction/division) and augmentation (random crops/horizontal flips)	0.954	-	@ajbrock	🔗 🔗
MobileNet	augmentation (horizontal flips)	0.950	-	@苏剑林	🔗
ResNet18	Normalization, random horizontal flip, random vertical flip, random translation, random rotation.	0.949	0.979	Kyriakos Efthymiadis	🔗
GoogleNet with cross-entropy loss	None	0.937	-	@Cenk Bircanoğlu	🔗
AlexNet with Triplet loss	None	0.899	-	@Cenk Bircanoğlu	🔗
SqueezeNet with cyclical learning rate 200 epochs	None	0.900	-	@snakers4	🔗
Dual path network with wide resnet 28-10	standard preprocessing (mean/std subtraction/division) and augmentation (random crops/horizontal flips)	0.957	-	@Queequeg	🔗
MLP 256-128-100	None	0.8833	-	@heitorrapela	🔗
VGG16 26M parameters	None	0.935	-	@QuantumLiu	🔗 🔗

Classifier	Preprocessing	Fashion test accuracy	MNIST test accuracy	Submitter	Code
WRN-28-10	standard preprocessing (mean/std subtraction/division) and augmentation (random crops/horizontal flips)	0.959	-	@zhunzhong07	🔗
WRN-28-10 + Random Erasing	standard preprocessing (mean/std subtraction/division) and augmentation (random crops/horizontal flips)	0.963	-	@zhunzhong07	🔗
Human Performance	Crowd-sourced evaluation of human (with no fashion expertise) performance. 1000 randomly sampled test images, 3 labels per image, majority labelling.	0.835	-	Leo	-
Capsule Network 8M parameters	Normalization and shift at most 2 pixel and horizontal flip	0.936	-	@XifengGuo	🔗
HOG+SVM	HOG	0.926	-	@subalde	🔗
XgBoost	scaling the pixel values to mean=0.0 and var=1.0	0.898	0.958	@anktplwl91	🔗
DENSER	-	0.953	0.997	@fillassuncao	🔗 🔗
Dyra-Net	Rescale to unit interval	0.906	-	@Dirk Schäfer	🔗 🔗
Google AutoML	24 compute hours (higher quality)	0.939	-	@Sebastian Heinz	🔗

Other Explorations of Fashion-MNIST

Fashion-MNIST: Year in Review

Fashion-MNIST on Google Scholar

Generative adversarial networks (GANs)

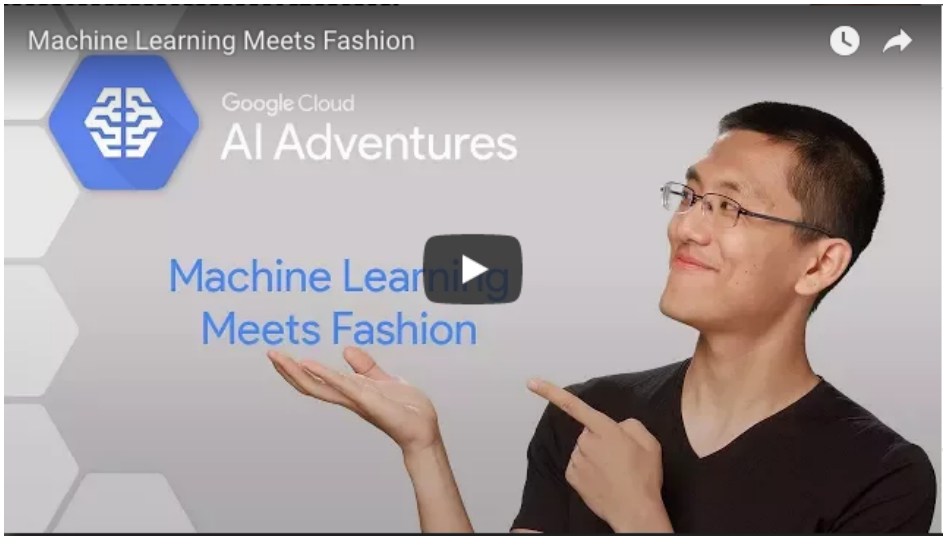
- Tensorflow implementation of various GANs and VAEs. (**Recommend to read!** Note how various GANs generate different results on Fashion-MNIST, which can not be easily observed on the original MNIST.)
- Make a ghost wardrobe using DCGAN
- fashion-mnist的gan玩具
- CGAN output after 5000 steps
- live demo of Generative Adversarial Network model with deeplearn.js
- GAN Playground - Explore Generative Adversarial Nets in your Browser

Clustering

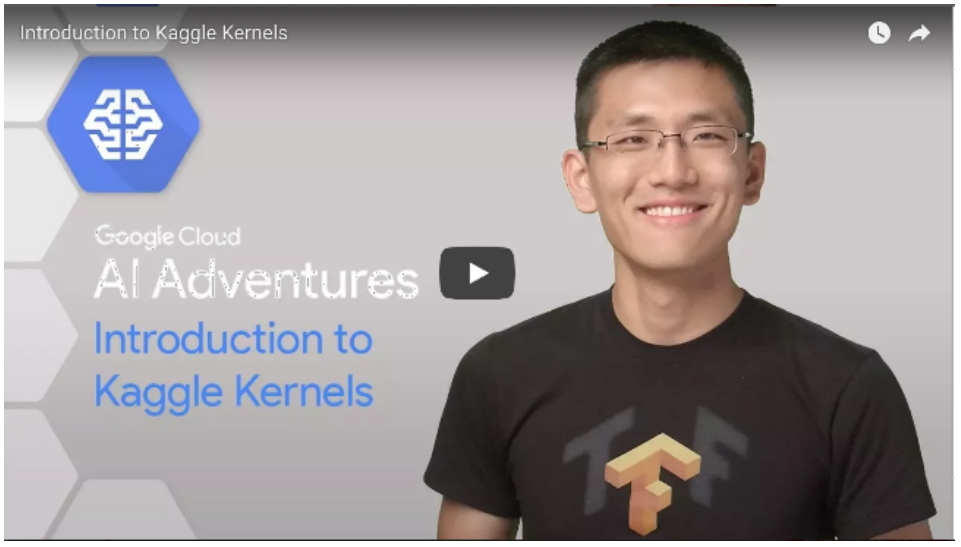
- Xifeng Guo's implementation of Unsupervised Deep Embedding for Clustering Analysis (DEC)
- Leland McInnes's Uniform Manifold Approximation and Projection (UMAP)

Video Tutorial

Machine Learning Meets Fashion by Yufeng G @ Google Cloud



Introduction to Kaggle Kernels by Yufeng G @ Google Cloud



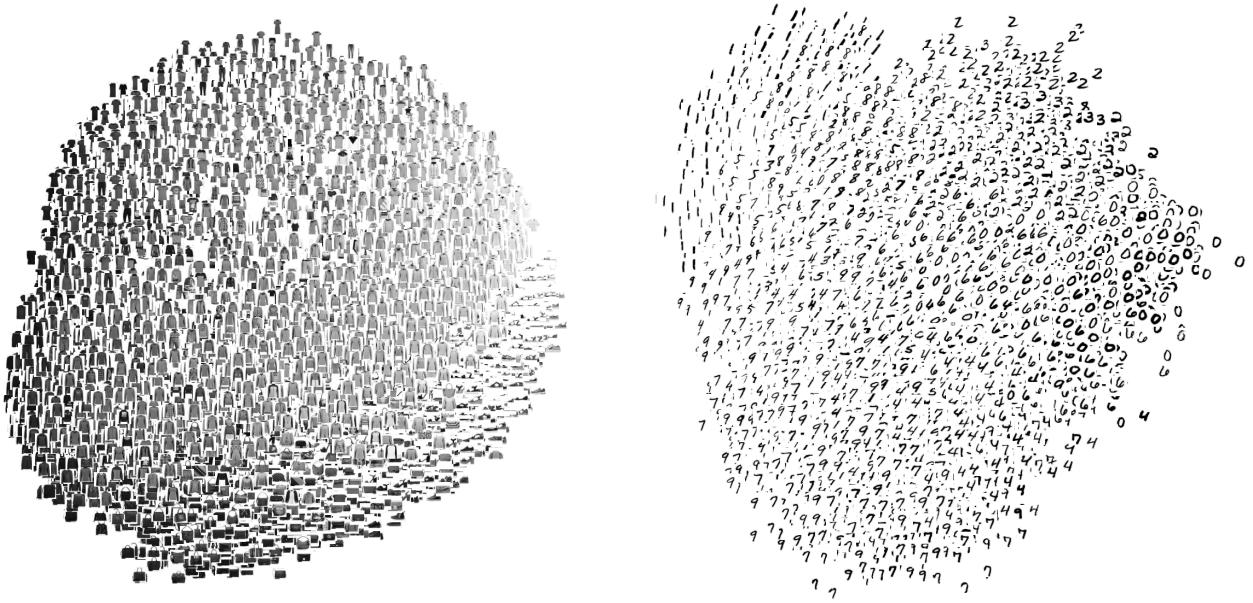
动手学深度学习 by Mu Li @ Amazon AI

Apache MXNet으로 배워보는 딥러닝(Deep Learning) - 김무현 (AWS 솔루션즈아키텍트)

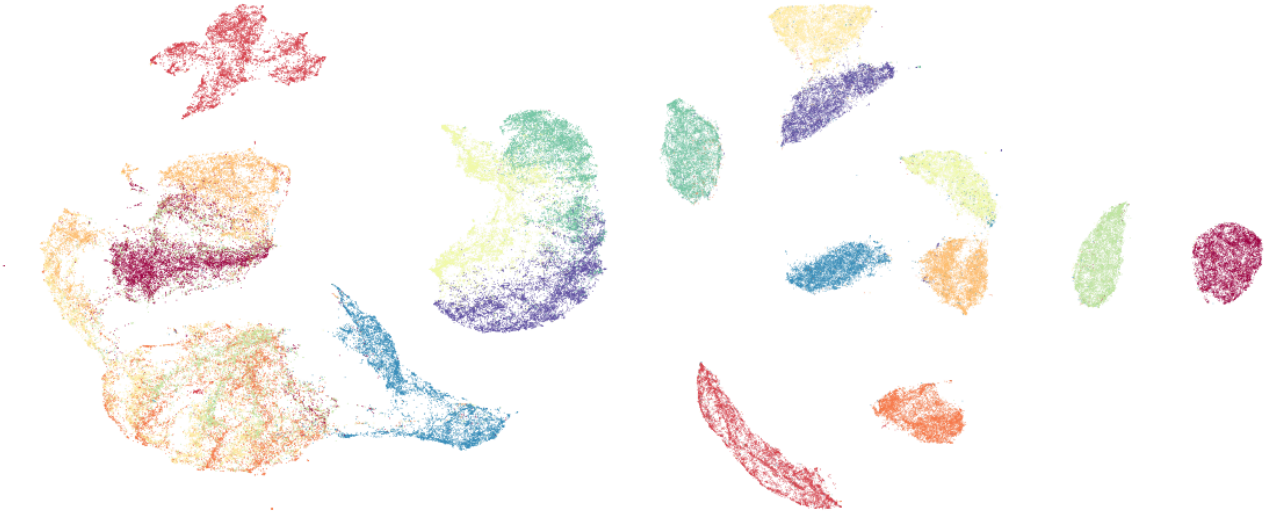
Visualization

t-SNE on Fashion-MNIST (left) and original MNIST (right)

PCA on Fashion-MNIST (left) and original MNIST (right)



UMAP on Fashion-MNIST (left) and original MNIST (right)



Contributing

Thanks for your interest in contributing! There are many ways to get involved; start with our contributor guidelines and then check these open issues for specific tasks.

Contact

To discuss the dataset, please use chat on gitter.

Citing Fashion-MNIST

If you use Fashion-MNIST in a scientific publication, we would appreciate references to the following paper:

Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. Han Xiao, Kashif Rasul, Roland Vollgraf. arXiv:1708.07747

Biblatex entry:

```
@online{xiao2017/online,
  author       = {Han Xiao and Kashif Rasul and Roland Vollgraf},
  title        = {Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms},
  date         = {2017-08-28},
  year         = {2017},
  eprintclass  = {cs.LG},
  eprinttype   = {arXiv},
  eprint       = {cs.LG/1708.07747},
}
```

Who is citing Fashion-MNIST?

License