



# INTELLIGENT SYSTEMS PROJECT REPORT

5 NOVEMBER, 2018

Promotion Neumann (5th year)

---

## Deep Learning with Fashion-MNIST

---

Vincent CANDAPPANE

Prof. Olivier BROUSSE

**University :** ESIREM

Software & Knowledge Engineering  
School,

9 Avenue Alain Savary,  
21000 Dijon, France



# Acknowledgments

This report has been prepared for the intelligent system module that has been done in ESIREM(Software & Knowledge Engineering School, Dijon) in order to study the practical aspect of machine-learning using Keras[3] with the purpose of fulfilling the requirements of the 5th year course of ESIREM.

I would like to express my sincere gratitude to our professor Mr. Olivier Brousse (R&D Manager at GlobalSensing Technologies) who have given his valuable time and his knowledge to understand the bases of Deep Neural Network and Keras API.

# Contents

	Introduction .....	1
<b>1</b>	<b>Fashion-MNIST dataset .....</b>	<b>2</b>
1.1	Introduction	2
1.2	Goal of the project	2
<b>2</b>	<b>Requirements .....</b>	<b>3</b>
<b>3</b>	<b>Architecture of Neural Network .....</b>	<b>4</b>
3.1	Steps to building our network	4
3.2	Importing the dataset	5
3.3	Setting network parameters	5
3.4	Model specification	6
3.5	Compiling & Backpropagation	7
3.5.1	Specify the optimizer .....	7
3.5.2	Specify the loss function .....	7
3.5.3	Compile .....	7
3.6	Data-augmentation	8
3.7	Learning Rate Manipulation	8
3.8	ModelCheckpoint : Save and reload	8
3.9	Fit : Putting it all together	8
3.10	Data visualization with TensorBoard	9
	Conclusion .....	10

## List of Figures

1.1	Fashion-MNIST dataset. . . . .	2
3.1	Architecture of the Neural Network . . . . .	6
3.2	Summary of last-layer activation function and loss function. . . . .	7
3.3	Final result : $val\_acc \rightarrow 94,2\%$ - $loss \rightarrow 17,95\%$ . . . . .	10
3.4	Data vizualisation with TensorBoard $\rightarrow val\_acc, loss$ . . . . .	10

# Introduction

Image classification is used in several applications, ranging from detecting object in images to recognizing life-threatening illnesses in medical scans.

In class, we have seen the MNIST dataset which is the most overused dataset for getting started with image classification. It contains **28x28 gray-scale images of handwritten digits**, each with an associated label indicating which number is written (an integer between 0 and 9).

The purpose of this report is to present the research and application work of a simple Deep Convolutional neural network of the Fashion-MNIST dataset using Tensorflow (Keras).

# 1. Fashion-MNIST dataset

## 1.1 Introduction

Researchers at Zalando (e-commerce company) have developed a new image classification dataset called Fashion-MNIST in hopes of replacing MNIST. This new image dataset is partially used for bench-marking machine learning algorithms for computer vision.

This dataset comprises **60,000 28x28 training images** and **10,000 28x28 test images**, including 10 categories of fashion products such as shirts, bags, coat, sandal, shoes, and other fashion items. Figure 1 shows all the labels and some images in Fashion-MNIST.

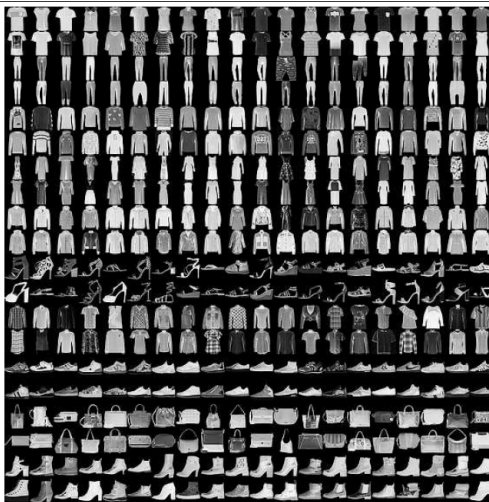
Label	Description	Examples
0	T-Shirt/Top	
1	Trouser	
2	Pullover	
3	Dress	
4	Coat	
5	Sandals	
6	Shirt	
7	Sneaker	
8	Bag	
9	Ankle boots	

Figure 1.1: Fashion-MNIST dataset.

## 1.2 Goal of the project

Our goal in this project is to create a Deep Convolutional Neural Network that would recognize correctly each of these items which respects the following performances :

- **% of success** : as much as possible (95% of success desired)
- **Number of parameters** : as low as possible
- **Code optimization** : Model optimization, inspiration model, choice of optimizer, early stopping, data augmentation, learning rate manipulation.

Finally, we will make an overall evaluation of the several approach developed during the project to combine a single model which respect the entire performance above (with higher accuracy rate).

## 2. Requirements

- **Python 3.5** : an interpreted high-level programming language for deep learning programming
- **Keras** : Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. It allows us as developers to quickly create models of neural networks and use a list of already existing datasets to train on, fashion mnist for instance.
- **Numpy** : NumPy is the fundamental package for scientific computing with Python.
- **Jupyter** : The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text.
- **Matplotlib** : Matplotlib is a Python 2D plotting library.
- **Tensorboard** : Suite of visualization tools from TensorFlow.
- **Tensorflow (gpu is recommended for optimal computation time)** : An open source machine learning framework.

For GPU support[4] : Install CUDA, CUBLAS, CUDNN then follow pre-requisites with tensorflow-gpu instead of tensorflow

## 3. Architecture of Neural Network

The focus of our neural network is to integrate the essential mechanisms for performing multi-label classification and generating class activation maps together. In class, we learn and manipulate 2 different efficient type of model for the image classification problem:

- MLP : multi-layered-perceptron
- CNN : convolutional-neural-networks

After several attempt of finding different solution, we noticed *CNN with 3 Convolutional Layers* was a good choice to learn image data in general. So it can be perfectly used for the Fashion\_MNIST dataset. The problem with that solution is the number of parameters which is really high and not recommended (approximately 10 millions parameters). My reasoning need to be completed with documentation more sophisticated to understand which type of model is efficient for which type of data. According to researches[1] from Jason Brownlee, we learn about several model such as :

- Basic Neural Networks
- Multi Layer Perceptrons (MLPs)
- Convolutional Neural Networks (CNNs with 2,3 and 4 convolutional Layers)
- VGG Like Model With Batch-normalization
- Recurrent Neural Networks (RNNs)
- Hybrid Network Models (HNMs)

The problem is as follows : all the model above have a specific advantage for our dataset. Per example, during the learning phase, MLP are more robust than CNN itself and that makes them theoretically better classifier in general. It is also known that MLP are best used when the input data comes from tabular datasets whereas CNN are best for image data. In theory, we can have interesting result by taking the best of both part, so it becomes an assembling of hybrid model composed with a MLP model followed by a CNN model.

### 3.1 Steps to building our network

While building a neural network, it's important to have a systematic approach. Here are the steps we will carry out while running our neural network :

1. Importing the dataset
2. Setting network parameters
3. Specify Architecture
4. Compile
5. Backpropagation
6. Fit : Putting it all together
7. Predict
8. Data Augmentation



## 9. Adjusting the Learning Rate

### 3.2 Importing the dataset

Since the main goal of Fashion MNIST is to serve as a drop-in replacement for the original MNIST, it can be imported in the same way MNIST is imported in Tensorflow. While importing the dataset, let's apply one-hot encoding to our class variables to convert them into a format that works better with machine learning algorithms.

```
import os
import keras
import numpy as np
from keras.datasets import fashion_mnist
from keras.models import Sequential, load_model
from keras.optimizers import Adam, Adadelta, RMSprop
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ModelCheckpoint, ReduceLROnPlateau, TensorBoard
from keras.layers import Dense, Dropout, Conv2D, MaxPooling2D, Flatten,
↳ Lambda, SpatialDropout2D

#----- Dataset of 60,000 28x28 training images & 10,000 28x28 test images.
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = np.reshape(x_train, (len(x_train), 28, 28, 1))
x_test = np.reshape(x_test, (len(x_test), 28, 28, 1))
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

#----- Convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
#-----
```

Now that the dataset imported, we can take a look at the shapes (dimensions) of both our training set and test set. It is important to get our matrix dimensions right before building the neural network, it can prevent computation errors in the future. The loss function used in the future is the 'categorical\_crossentropy', so it is important to convert class vectors to binary class matrix.

### 3.3 Setting network parameters

First, let's declare the above mentioned parameters which describe our network. This will simplify hyper-parameter tuning during later phases.

```
epochs = 25
num_classes = 10
batch_size = 300
input_shape = (28, 28, 1)
filepath = "cnn_model_best.hdf5"
```

### 3.4 Model specification

In this part we setting up the layers in the model, implementing the architecture of the CNN learned in class. For more feature extraction, we assemble a second CNN on top of the previous one. The [if-else] condition is used for gaining time in terms of processor computation : if a '.hdf5' exist, it will directly load the previous weights without the need of relearning from the beginning. Thus, it will tend to achieve greater accuracy quickly. Here the final Architecture of the Neural Network using TensorBoard :

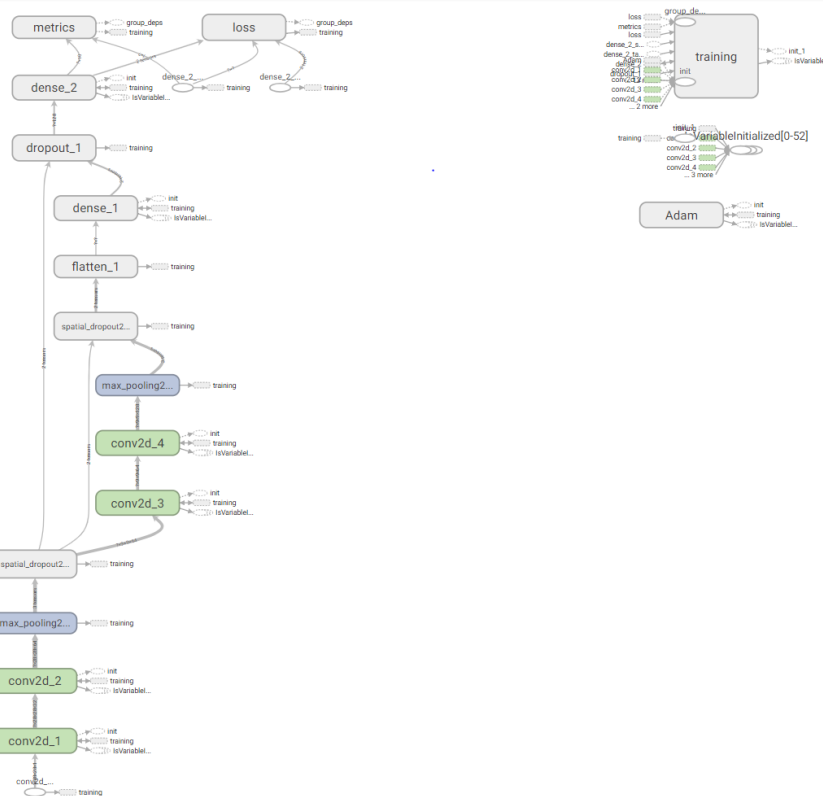


Figure 3.1: Architecture of the Neural Network

#### → First CNN

- 2D convolutional layer of 32 and a kernel-size of (3,3) with the input shape of (28, 28, 1).
- 2D convolutional layer of 64 and a kernel-size of (3,3)
- 2D maxpooling layer with a pool size of (3,3) → The max-pooling layer is an operation for spatial data, it is used to reduce the data preventing the model to be extrapolated too much.
- 2D SpatialDropout of 0,25 → This version performs the same function as Dropout, however it drops entire 2D feature maps instead of individual elements.

#### → Second CNN

- 2D convolutional layer of 64 and a kernel-size of (3,3)
- 2D convolutional layer of 128 and a kernel-size of (5,5)
- 2D max-pooling layer with a pool-size of (3,3)
- 2D Spatial dropout of 0,25.

#### → Ending with a MLP

- Flatten() : Conversion into 2D data to 1D. Flattening the data help to fit into the MLP.
- Dense(128, activation='relu') : dense layer of 128 → Dense layers connect everything.
- dropout of 0,5 : Drop only the individual element despite
- Dense layer of 10, corresponding to the 10 possible image category.

## 3.5 Compiling & Backpropagation

### 3.5.1 Specify the optimizer

The optimizer regulate how the learning process works and how the learning rate is supposed to be set. Backpropagation is an essential step for our neural network, because it decides how much to update our parameters (weights and biases) based on the cost from the previous computation. There is many mathematical complex options as follows :

- **SGD** : Stochastic gradient descent optimizer. Includes support for momentum, learning rate decay, and Nesterov momentum.
- **RMSprop** : It is recommended to leave the parameters of this optimizer at their default values (except the learning rate, which can be freely tuned). This optimizer is usually a good choice for recurrent neural networks.
- **Adagrad** : Adagrad is an optimizer with parameter-specific learning rates, which are adapted relative to how frequently a parameter gets updated during training. The more updates a parameter receives, the smaller the updates.
- **Adam** : Default parameters follow those provided in the original paper. There is other variant of Adam : Adamax, Nadam.

After testing those different optimizer into the fashion\_mnist dataset, the final optimizer chosen for this project is the default Adam for backpropagation with a predefined learning rate(lr=0.001).

### 3.5.2 Specify the loss function

Problem type	Last-layer activation	Loss function
Binary classification	sigmoid	binary_crossentropy
Multiclass, single-label classification	softmax	categorical_crossentropy
Multiclass, multilabel classification	sigmoid	binary_crossentropy
Regression to arbitrary values	None	mse
Regression to values between 0 and 1	sigmoid	mse or binary_crossentropy

Figure 3.2: Summary of last-layer activation function and loss function.

For multi-class classification, the last layer in the model uses a softmax function for class prediction. The use of "softmax" unlock the possibility of the model to choose between multiple values which one is the highest corresponding to the predicted class.

And the training process uses a 'categorical\_crossentropy' function as the loss function[2]. The loss function chosen for this project is as follows : 'categorical\_crossentropy' is similar to log loss(Lower is better).

### 3.5.3 Compile

Output layer has separate node for each possible outcome, and uses 'softmax' activation. To compile step for easy-understand diagnostics, we use *metrics* = ['accuracy'] :

```
model.compile (loss = keras.losses.categorical_crossentropy, optimizer =
↳ keras.optimizers.Adam(lr=0.001), metrics=['accuracy'])
model.summary()
```

### 3.6 Data-augmentation

In order to make the most of our few training examples, we will "augment" them via a number of random transformations, so that our model would never see twice the exact same picture. This helps prevent over-fitting and helps the model generalize better. In Keras this can be done via the *keras.preprocessing.image.ImageDataGenerator* class. This class allows you to:

- configure random transformations and normalization operations to be done on your image data during training.
- instantiate generators of augmented image batches (and their labels).

```
#----- Data Augmentation
gen = ImageDataGenerator(horizontal_flip=True, zoom_range=0.15,
↪ shear_range=.1, fill_mode='nearest')
```

**Observations :** All the models used during this project achieved a higher accuracy after using data augmentation. The data augmentation creates new training samples by rotating horizontally(not vertically based to the fashion\_mnist dataset), shifting and zooming on the training samples.

### 3.7 Learning Rate Manipulation

Learning rate schedules seek to adjust the learning rate during training by reducing the learning rate according to a pre-defined schedule. If in an interval of 5 epochs, the model stop learning and the loss wouldn't change, ReduceLROnPlateau() function will decrease automatically on a factor of 0,2.

```
#----- Reduce Learning Rate Function
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5,
↪ verbose=1, mode='auto', min_lr=0.00000001)
```

### 3.8 ModelCheckpoint : Save and reload

ModelCheckpoint save the model after every epoch. We set ModelCheckpoint to save only the best model, so the previous recovery model is replaced by the next best model. If there is no more best, nothing is saved after the previous 'best' model.

```
#----- Checkpoint
checkpointer = ModelCheckpoint(filepath=filepath, verbose=1,
↪ monitor="val_acc", save_best_only=True)
```

### 3.9 Fit : Putting it all together

Finally, let's combine all our previously-created functions into one single function called model(). Here we are combining 'fit' and 'fit\_generator' together, the first 'fit' learn the normal training images and the second 'fit\_generator' learn in addition of the normal images, the modified images via data-augmentation.

All the models are initially trained for 25 epochs and another 25 epochs with lowering the learning late in the case if the 'accuracy value' is not changing for 5 complete epochs.

The learning rate is decreasing with a factor of 0,2. After the initial 25 epochs, we added data augmentation, which generates new training samples by rotating, shifting and zooming on the training samples, and trained for another 25 epochs. This process is repeated 5 times to increase the accuracy at his optimal rate.

```
for i in range(0, 5):
    #----- Fit
    model.fit(x_train,
              y_train,
              epochs=epochs,
              batch_size=batch_size,
              verbose=1,
              shuffle=True,
              validation_data=(x_test, y_test),
              callbacks=[tensorboard, checkpointer, reduce_lr])

    #----- Reinitilize learning rate & Reload best model
    model.compile(loss=keras.losses.categorical_crossentropy,
                  optimizer=keras.optimizers.Adam(lr=0.001), metrics=['accuracy'])
    model = load_model(filepath)

    #----- Fit Generator
    batches = gen.flow(x_train, y_train, batch_size=batch_size)
    model.fit_generator(batches,
                        steps_per_epoch=len(x_train)/batch_size,
                        epochs=epochs,
                        validation_data=(x_test,y_test),
                        verbose=1,
                        shuffle=True,
                        validation_steps=10000//batch_size,
                        callbacks=[tensorboard, checkpointer, reduce_lr])

    #----- Display last iteration result
    print('Test loss:', score[0])
    print('Test accuracy:', score[1])
```

After building the network and training it, the function will compute the model's accuracy, and return the final updated parameters dictionary.

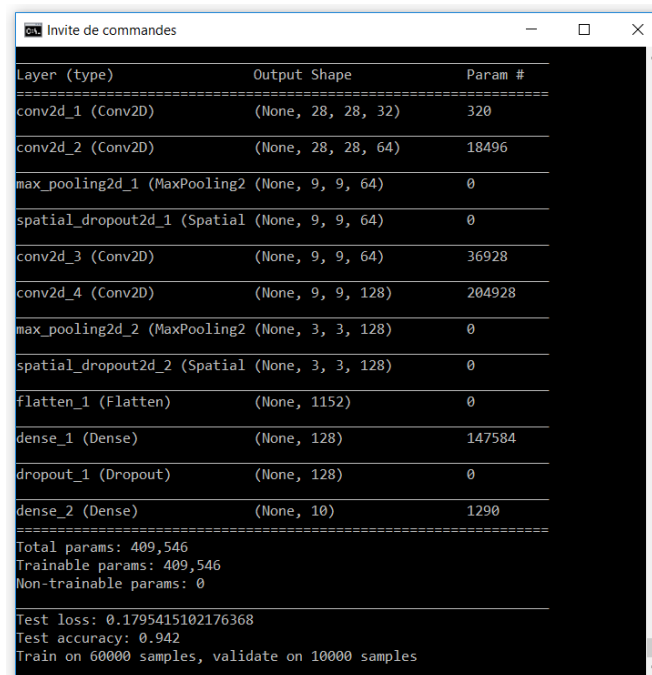
### 3.10 Data visualization with TensorBoard

TensorBoard is a utility from TensorFlow that allows you to visualize data and how it behaves. During the project every data was collected in a single file called 'logs'. Here how to implement a TensorBoard server :

```
#----- Tensorboard
tensorboard = TensorBoard(log_dir='./logs', histogram_freq=0, batch_size=32,
                           write_graph=True, write_grads=False, write_images=False,
                           embeddings_freq=0, embeddings_layer_names=None, embeddings_metadata=None,
                           embeddings_data=None) \\\
```

# Conclusion

This report presents a study on deep learning for Fashion-MNIST in the context of multi-label classification. As a result of the final model using Convolutional Neural Network, we obtain a `val_accuracy` of 94,2% for using less than 410k (409,546 parameters used), it is likely near 95%). Another approach like the DenseNet can be another method for the Fashion-MNIST dataset.



```
Layer (type)                Output Shape                Param #
-----
conv2d_1 (Conv2D)           (None, 28, 28, 32)         320
conv2d_2 (Conv2D)           (None, 28, 28, 64)         18496
max_pooling2d_1 (MaxPooling2 (None, 9, 9, 64)          0
spatial_dropout2d_1 (Spatial (None, 9, 9, 64)          0
conv2d_3 (Conv2D)           (None, 9, 9, 64)           36928
conv2d_4 (Conv2D)           (None, 9, 9, 128)          204928
max_pooling2d_2 (MaxPooling2 (None, 3, 3, 128)         0
spatial_dropout2d_2 (Spatial (None, 3, 3, 128)         0
flatten_1 (Flatten)         (None, 1152)                0
dense_1 (Dense)             (None, 128)                 147584
dropout_1 (Dropout)         (None, 128)                 0
dense_2 (Dense)             (None, 10)                  1290
-----
Total params: 409,546
Trainable params: 409,546
Non-trainable params: 0

Test loss: 0.1795415102176368
Test accuracy: 0.942
Train on 60000 samples, validate on 10000 samples
```

Figure 3.3: Final result : `val_acc` → 94,2 % - `loss` → 17,95 %

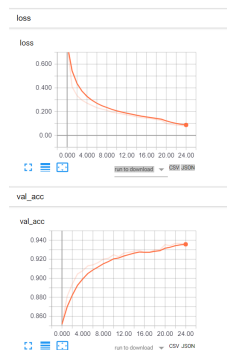


Figure 3.4: Data vizualisation with TensorBoard → `val_acc`, `loss`

## Bibliography

- [1] Jason Brownlee. *When to Use MLP, CNN, and RNN Neural Networks*. URL: <https://machinelearningmastery.com/when-to-use-mlp-cnn-and-rnn-neural-networks/>.
- [2] Losses Functions. *A loss function (or objective function, or optimization score function)*. URL: <https://keras.io/losses/>.
- [3] Adrian Rosebrock. *Building a simple Keras + deep learning REST API*. URL: <https://blog.keras.io/index.html>.
- [4] TensorFlow. *TensorFlow GPU supports*. URL: <https://www.tensorflow.org/install/gpu>.