M.Sc. in Data Science

- Course: Deep Learning
- Instructor: Prof. P Malakasiotis
- Author: Andreas Stavrou (f3352120)
- Assignment 1

Introduction and scope

Homework description

Submit a report (5 - 10 pages, PDF format) for the following machine learning project. Explain briefly in the report the architectures that you used, how they were trained, tuned, etc. Describe challenges and problems and how they were addressed. Present in the report your experimental results and demos (e.g., screenshots) showing how your code works. Do not include code in the report but include a link to a shared folder or repository (e.g., in Dropbox, GitHub, Bitbucket) containing your code. The project will contribute 40% to the final grade.

Given an image of a fashion item, build a deep learning model that recognizes the fashion item. You must use at least 2 different architectures, one with MLPs and one with CNNs. Use the Fashion-MNIST dataset to train and evaluate your models. More information about the task and the dataset can be found at https://github.com/zalandoresearch/fashion-mnist. The dataset is also available from Tensorflow and Keras.

Goals Description

The scope of the assignment is to lead the student to a firm understanding of key tasks related to creating, training and evaluating the performance of some neural network architectures, specifically *MLP*s and *CNN*s, for the task of image classification. The assignment is not designed with the end goal to produce the best classifier, rather to allow the student to demonstrate techniques to tweak the performance of neural networks.

With this goal in mind, the assignment will focus on the following tasks:

- Specifying the task at hand: this is an essential step which, given the nature of the problem, will define its parameters, such as loss function to use.
- Ingestion and preprocessing of input data so as to be in the correct input form for the neural network architectures that will be developed.
- Sound, both theoretically and practically, splitting of train / dev / test sets.
- Sound definition of the metrics required to assess the model performance.
- Implementation and presentation of a few *MLP* and *CNN* architectures, their relative capacity, performance and possible shortcomings. Description of the weights initialization, activation functions, regularization techniques and optimization algorithm selected will also be provided and assesed.
- Hyperparameter tuning for the best architecture selected, after manual implementation and assesment.

Setup and environment

Deep learning is well-known to benefit from *CUDA*-accelerated infrastructure, since matrix operations are orders of magnitude faster on GPU than CPU hardware. The development machine is equiped with an *M1 Chip*. This will result in some training and tuning limitations but the task at hand was not to solve the problem, rather than successfully tackle it.

Specifying the task

The task of this assignment is to preform multiclass classification on the input set, based on the training set labels provided.

Data ingestion

Initial dimension of ingested data are as follows:

Dataset	Value
X_train	(60000, 784)
y_train	(60000,)
X_test	(10000, 784)
y_test	(10000,)

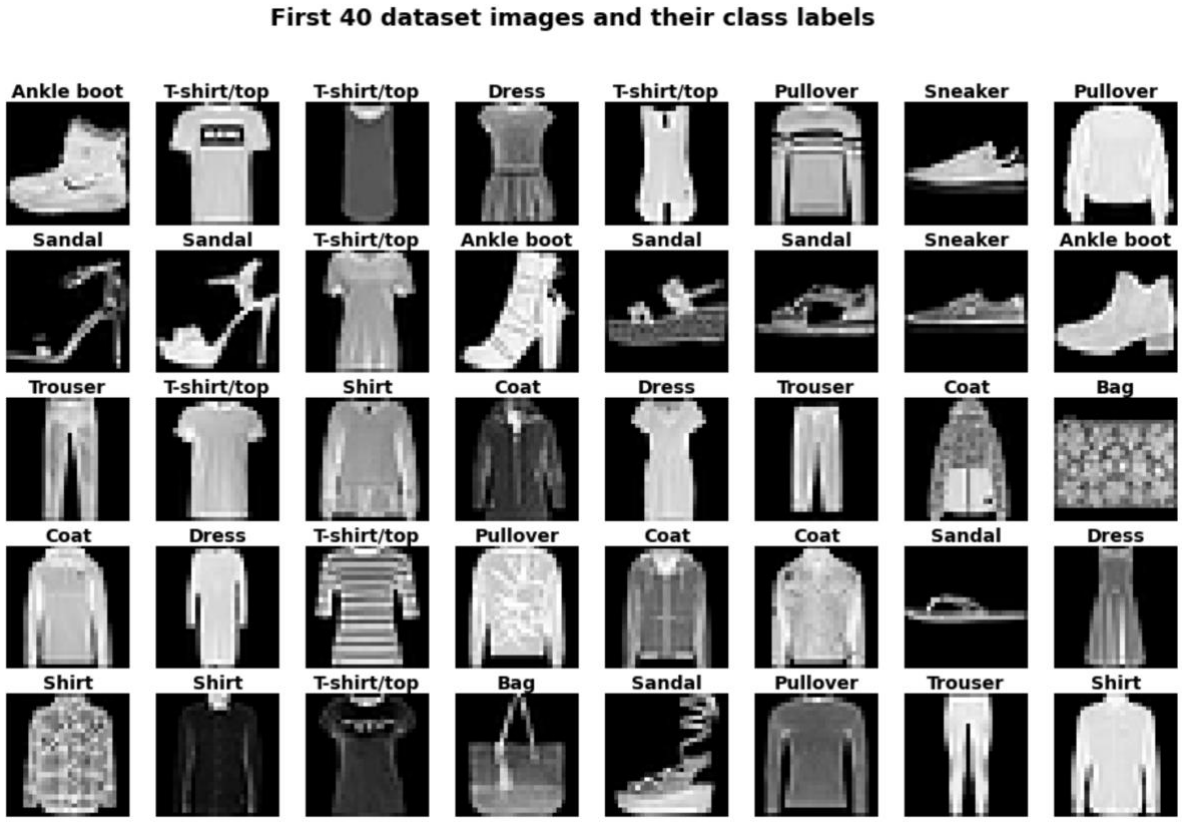
We observe that there exist training examples and test examples. The data features are rolled-out in arrays of shape, being the dimensions of input images.

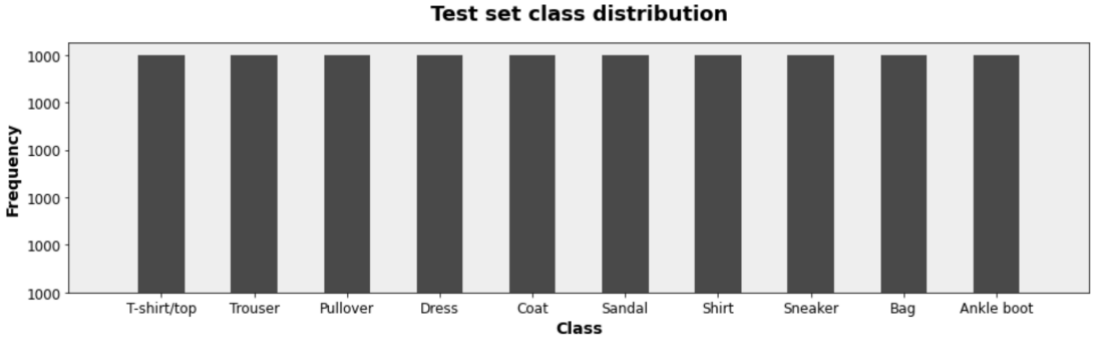
Data augmentation

Data augmentation, which is a technique to increase the diversity of your training set by applying random transformations was used through Keras image preprocessing package *ImageDataGenerator.* The dataset was adequate for the initial approach to the problem; however, a particular class ("Shirt") was often misclassified and had the lowest accuracy. To tackle that issue, I tried to ingest more data to the model by rotating, applying zoom and shifting horizontally and vertically. The above resulted in an increase of the accuracy of that class, which went from 0.78 f1-score to 0.83 f1-score.

EDA

Brief exploratory data analysis follows below and focuses mainly on getting a feel of the data, as well as making sure that no class imbalance problem exists.





We observe that the distribution of classes in both the training and the test set are uniformly distributed, hence the dataset does not present a class imbalance problem.

Data preprocessing

Pixel values normalization

The purpose of normalization is to fit the feature space to because unscaled input variables can result in a slow or unstable learning process. Normalization was achieved simply by dividing each pixel value by 255 (the maximum pixel value for grayscale images).

Splitting data into train / validation / test sets

Since our dataset has already been split to train / test sets (*(X_train, y_train)*, *(X_test, y_test)* respectively), we shall proceed further by retaining a percentage (0.1) of the test data for creating a *validation* set. The *validation* set will be used for tuning hyperparameters of the model architectures, so as to acquire a robust model for inference.

We shall also be extra careful to retain the uniform nature of the target class distribution so as not to introduce bias. To do this, we shall split the sets in a stratified fashion.

General remarks / architectural choices

Commenting on the neural net depth, breadth and architecture (*MLP*s vs. *CNN*s), how to we know which size of network to create? A *MLP* with at least a hidden layer and a non-linearity is a universal continuous function approximation. However, creating a shallow *MLP* is trying to model the function in low vector space. Empirical observations show that deeper neural architectures work better in real problems. In practice it is often the case that 3-layer *MLP*s will outperform 2-layer *MLP*s, but going even deeper (4, 5, 6-layer) rarely helps much more.

This is in stark contrast to *CNN*s for the task of image classification, where depth has been found to be an extremely important component for a good recognition system (e.g. on order of learnable layers). One argument for this observation is that images contain hierarchical structure so several layers of processing make intuitive sense for this domain.

Recall that model capacity is the ability of our model to optimaly represent the problem at hand. A lower capacity model with fail to fit its parameters during training time (i.e. the model will *underfit* the data), while an excesive capacity model will fail to generalize (i.e. it will *overfit* to the training instances)

Loss function

We are dealing with a multiclass classification task. For this reason, the appropriate loss function to use, regardless of network architecture (*MLP* or *CNN*), is **categorical cross-entropy** which is defined as:

$$J(\mathbf{w}) = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right]$$

where:

w: the model parameters
$\mathbf{y_{i:}}$ the true class label
$\mathbf{y_{i\ (hat)}}$**:** the predicted class label
N: the number of classes

Output layer activation function

Since our task is a *multi-class classification problem*, the activation function of the output layer is a *softmax*.

Batch size

Empirical observation leads to the conclusion that *batch_size* affects overall training performance. A very high *batch_size* will affect model accuracy, while a very small batch_size will have a negative impact on convergence. After experimentation with different batch sizes, I choose 128.

Regularization

The predominant method of regularization in neural net architectures is the introduction of *dropout* layers in between the hidden layers of the architecture. *Dropout* layers enforce random non-selection of weights, up to a configurable percentage of all weights it the current layer (i.e. a hyperparameter), thereby making sure that the model does not learn some parameters at random, effectively avoiding overfitting.

Hyperparameter search / optimization

In contrast to classic machine learning models, one major challenge that one faces while working with deep learning is that there are lot of parameters to hyper tune. Hence it becomes important to appropriately select correct parameters so as to avoid overfiting and underfitting.

To tackle the hyperparameter space search problem, I used the Keras tuner library and to be more specific the Hyperband option for hyperparameter tuning.

MLPs

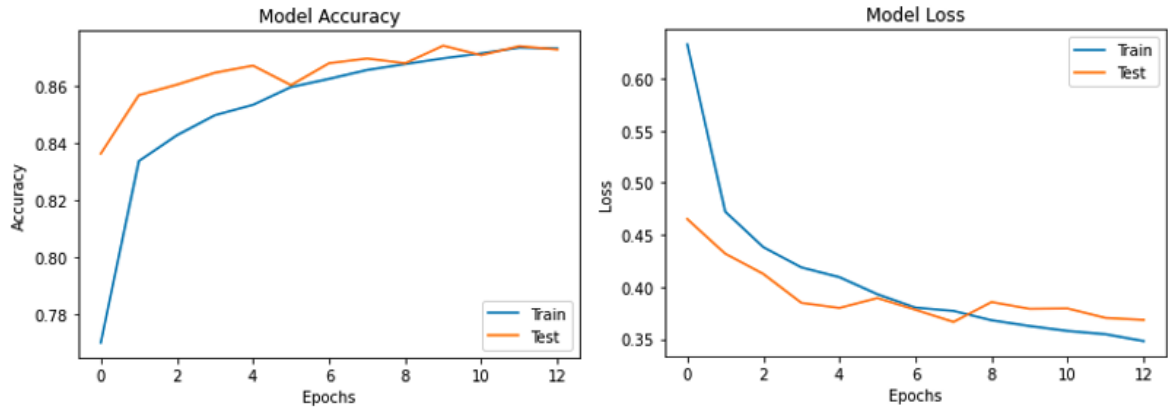
The first architecture decision about the MLP was the network's depth and breadth. I created 3 MLPs with the following architectures / characteristics to try and find the best candidate for further exploration.

- A small MLP (3 layers),
- A medium MLP (5 layers),
- A large MLP (7 layers), with the following architectures:

The next manual improvement on those baselines architectures where measures to avoid overfitting, namely regularization and early stopping. Regularization is achieved by introducing *dropout* layers in the MLP architecture. *Early stopping* monitors *loss*, *val_loss* for identyfying a point in time (epoch) in which *val_accuracy* hasn't decreased w.r.t. previous epochs.

The results obtained after training for 100 epochs, using *Adam* as our optimizer with parameters learning rate=0.001, and regularization are as follows:

Architecture	Test loss	Test accuracy
mlp_small	0.6279	**0.8906**
mlp_medium	0.4144	0.8856
mlp_large	**0.3807**	0.8710



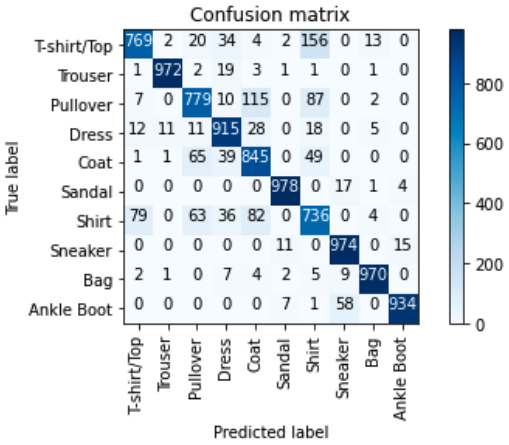
Accuracy and loss results on 7-layer MLP

Hypertuning

For the hyper tuning of the MLP I used the following search space:

- Number of layers: 2,3,4,5,6
- Number of units: 32,64,128,256,512
- Dropout: 0, 0.1, 0.2, 0.3
- Learning Rate: 0.01, 0.001, 0.0001

which yields a test accuracy of 0.8932.



Confusion matrix results on 7-layer MLP

CNNs

Convolutional neural networks are a flavor of neural nets tailored for the task of image of performing deep learning on input data that are, usually, images. This assumption allows us to encode certain properties into the architecture.

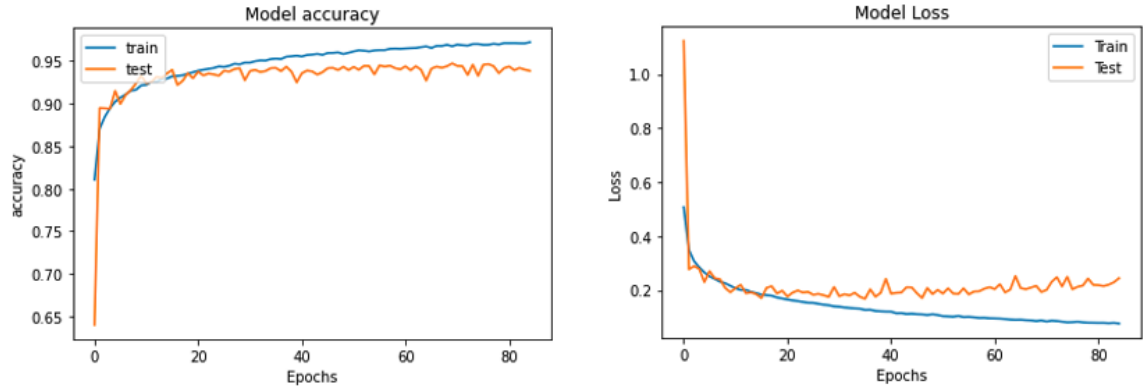
During my experimentation on trying to find the best CNN model for the task, I can pinpoint 5 different separate stages that I followed to inclemently improve my model's accuracy.
The ***stage 1*** was to build a **baseline CNN model** to get a feel of the performance. Then, the ***stage 2*** was to **reflect on the dataset** and find ways to maybe improve the performance by altering some images, as done with data augmentation. After that, the ***stage 3*** was to **experiment with different CNN architectures** regarding depth and breadth. Following that, the ***stage 4*** was to **hyper tune the parameters** of my empirical best architecture. Stage 3 and 4 could be done in one step if I had more computational recourses so that I could try the Keras tuner with all available options such as layers, dropouts, kernel sizes, pool sizes, filters, all at once. Finally, the ***stage 5*** was that after I got the best results from hyper tuning, I got that model architecture and parameters and created 15 CNN models of that design to later implement an **ensemble CNN** to vote for the right classification.

For the CNN I tried two different models, both medium sized with 3 and 4 convolutional blocks each, with the difference that the first model had 315k parameters whereas the second model had almost double that at 872k parameters.

I manually improved the performance of both by adding dropouts and batch normalization layers. For the first model I tried to mimic the LeNet 5 CNN architecture with some minor changes regarding the regularization layers. The second model was the same with an extra convolutional block.

The results obtained after training for 100 epochs, using *Adam* as our optimizer with parameters learning rate=0.001, and regularization are as follows:

Architecture	Test loss	Test accuracy
4_block_cnn	0.1952	0.9308
5_block_cnn	0.2138	**0.9409**



Accuracy and loss result for 5 Block CNN with Adam(lr=0.001)

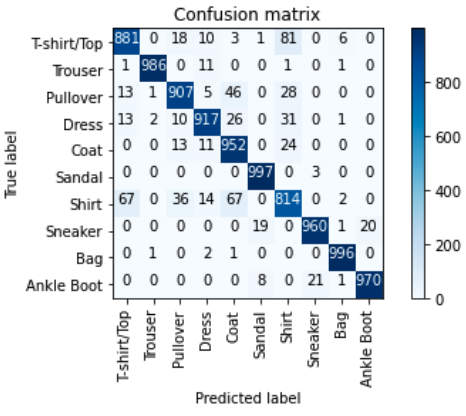
For the hyper tuning of the 5_block_cnn I used the following search space:

- Filters Layer 1: 16,32
- Kernel: 3,5
- Pool size: 2,4
- Filters Layer 2,3,4: 32,64,128
- Dropout: 0.0-0.5
- Learning rate: 0.0001-0.01

The results after the tuning were about the same and slightly worse than those of the best model, maybe because of the different data split.

Classification Report and confusion matrix

	precision	recall	f1-score	support
T-shirt/Top	0.90	0.88	0.89	1000
Trouser	1.00	0.99	0.99	1000
Pullover	0.92	0.91	0.91	1000
Dress	0.95	0.92	0.93	1000
Coat	0.87	0.95	0.91	1000
Sandal	0.97	1.00	0.98	1000
Shirt	0.83	0.81	0.82	1000
Sneaker	0.98	0.96	0.97	1000
Bag	0.99	1.00	0.99	1000
Ankle Boot	0.98	0.97	0.97	1000
accuracy			0.94	10000
macro avg	0.94	0.94	0.94	10000
weighted avg	0.94	0.94	0.94	10000



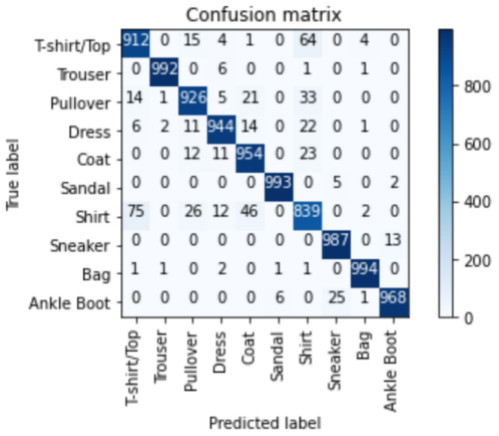
As the last stage for improvement, I wanted to train 15 models with the best CNN model architecture and then have them vote for the classification task as part of an ensemble CNN. The above ensemble of 15 CNN models scored as shown:

Architecture	Test loss	Test accuracy
Ensemble_CNNs	0.1552	**0.9509**

The detailed results are as follows:

Classification Report and confusion matrix

	precision	recall	f1-score	support
T-shirt/Top	0.90	0.91	0.91	1000
Trouser	1.00	0.99	0.99	1000
Pullover	0.94	0.93	0.93	1000
Dress	0.96	0.94	0.95	1000
Coat	0.92	0.95	0.94	1000
Sandal	0.99	0.99	0.99	1000
Shirt	0.85	0.84	0.85	1000
Sneaker	0.97	0.99	0.98	1000
Bag	0.99	0.99	0.99	1000
Ankle Boot	0.98	0.97	0.98	1000
accuracy			0.95	10000
macro avg	0.95	0.95	0.95	10000
weighted avg	0.95	0.95	0.95	10000



Looking at the precision, recall, f1-score of each class, we can see a slight improvement in all classes. The shirt, which was the weakest class in classifing, also gained a 3% f1-score increase which is more than any model we tried through hypertuning.

Implementation

All the implementation for the above can be found on: [Google Drive]

Conclusion

With this assignment I found that the CNN models perform better than the MLP. The CNN through careful parameter tuning, managed to show its ability on image classification. For future work I would like to further hyper tune the model and also try out different optimizers. Transfer learning with Resnet and VGG did not manage to score better than our results however I would like to further experiment on that because on more complicated tasks I believe those architectures are going to outperform the other CNN models.