École Polytechnique Fédérale de Lausanne
EPFL-2024 - Machine Learning

Report - Project

Étudiants:
BRUNO Elie
THONET Mateo
CHARVILLAT Lancelot

Professeur:
Mathieu SALZMANN

December 5, 2024

Contents

1 Milestone II

1.1 Introduction

1.1.1 Enhancements to Project Structure and Codebase

To streamline our model development process and enhance code readability and maintainability, we decided to implement the MLP and the CNN classes with two distinctions, we can pass an array to the class and it will automatically create the hidden layers. It makes cross_validation or just different tests easier to do and we therefore do not need to hard code any values. We also implemented acceleration regarding tasks, it implies the following, you can use the program argument `--device` to select what type of acceleration you want depending on your system (mps/nvidia GPU/CPU). It made our code extremely faster since CNN's and Transformer were slow.

A 20% validation set was included in main. From there we wanted to print different outputs if the `--test` was given or not. So we changed the lines that were computing accuracy's and F1 scores to make them clearer.

1.2 Data Processing

First the 28x28 images are flattened into a vector. Then, the data is normalized before training, to equalize their contributions by transforming each feature x using:

$$x_{\text{normalized}} = \frac{x - \mu}{\sigma}$$

where μ and σ are the feature's mean and standard deviation. This step prevents features with larger scales from dominating the model.

1.2.1 K-Fold Validation

To better tune our hyper parameters, we decided to use a 3-fold cross validation. That way, less data is lost compared to the standard validation set. The optimal parameter is chosen respectively for each method, by finding the parameter (from a large set of predefined values) which yields the highest validation metric chosen for the task and the model.

1.3 Learning rate & Loss function & Epochs

For our 3 models, on top of varying architectures we try different learning rates over [0.1, 0.01, 0.001, 0.0001]. We found the best learning rate to 0.01. For our 3 models, the cross entropy loss function is used. The model are runned on 100 epochs.

1.4 MLP

1.4.1 Method

For each hidden unit a bias term is added before activation. For the activation of hidden layers, the ReLu function was used, as for the activation of the output a softmax. To evaluate the model performance we use the macro F1 score. We use the SGD optimization algorithm.

1.4.2 Experiment and Results

To determine the architecture of our model, we use a grid-search type of optimisation iterating over the depth and number of neurons per layer. We tried varying between 1 and 3 hidden layers and between 64, 128, 256, and 512 per layer**??**. We found that the best set of parameters are [512, 256].

	Training Set	3 Cross Validation Set	20% Validation Set
MacroF1	0.86	0.828	0.83
Accuracy	86	82.9583	84

Table 1: *MacroF1 and accuracy for [512, 256]*

1.4.3 Discussion

As we can see we reach an accuracy of 84% and macro f1 0f 0.83 on the training set, indicating that our model has a certain capacity to generalize. The runtime is of roughly 75 seconds which is quite fast. However, the performance of the model is not the best.

1.5 CNN

1.5.1 Method

Same as MLP for the beginning however instead of SGD we used Adam, see 1.4.1. For padding, if the kernel is of size k then the features map will be reduced to k-1, therefore each side will padded with (k-1)/2 zeros.

1.5.2 Experiment and Results

As we choosed CNN for the contest, this section have more material than the others. The choice of Adam for the optimizer was decided after trying many of them: Nadam, AdamW, SGD, SGD with momentum,RMSprop, Adam with weight decay. however Adam was the one to stand out with a small difference of 0.3% in the validation set. We also tried different activation function, such as Mish, SiLU, Leaky Relu, tanH and obviously ReLU. Most of them weren't as effective as ReLU (down 1% in the validation set) yet Mish was on par with it, But we decided to stick to ReLU as it was the most commonly used. For the achitecture of our model, we tried to increase complexity gradually. First with two convolutional layers with convolutional channels [16, 32] then three [32, 64, 128], with each layer followed by a max pooling reducing size by 2 and with 2 fully connected layers. We found that the best results were obtained by the second architecture [32, 64, 128].

	Training Set	3 Cross Validation Set	30% Validation Set
MacroF1	0.998	0.8897	0.896
Accuracy	99.798	89.013	89.633

Table 2: *MacroF1 and accuracy for [32, 64, 128]*

To then rank second in the competition, we started looking at more advanced CNN's architecture, we discovered the Dropout method to prevent over fitting early on in the training process. As of now, we have the following accuracies

	Training Set	3 Cross Validation Set	20% Validation Set
MacroF1	1	0.928	0.937
Accuracy	99.996	92.7283	93.667

Table 3: *MacroF1 and accuracy for our best CNN architecture*

We also tried other type of CNN, such as the ResNet. Used to remove the vanishing gradient issue in deep neural network, we thought that it was a good idea to optimize our model. We however didn't had enough knowledge to implement it correctly and ended up sticking with our current architecture To approve our analysis we have the following confusion matrix and model summary. table 4

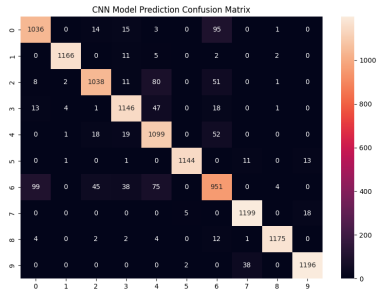

Figure 1: Confusion Matrix of our CNN model

We clearly see what is wrong with our predictions we are confusing classes, 0,2,3,4 and 6, by checking the labels of the dataset it confirms our analysis since those are similar clothing.

1.5.3 Discussion

We observe very high macro f1 and accuracy on the test set and a roughly 10% decrease to the validation set. This indicates that our model has a tendency to overfit. However, the accuracy and f1 score on the validation set are still at a higher level than for the MLP, but this comes at a large cost of roughly 650 seconds to run for a basic model. Since we had to try bigger models for the contest, we choosed to go to colab entreprise and use V100 GPUs to make our testing easier. We created a jupyter notebook independant from our current code where we tried multiple ideas, hence the code used for the contest might be a bit different from the one submitted.

1.6 Vision Transformer

1.6.1 Patchification

In the context of image recognition we can't really divide our text into small words so we divide our image into equal sized patches followed for each one by a linear projection.

1.6.2 Positional Encoding

So that each token captures its position in the image we use sinusoidals to encode location:

$$\mathrm{PE}(pos, 2i) = \sin(\frac{pos}{10000^{2i/d_{\text{model}}}})$$

$$\mathrm{PE}(pos, 2i+1) = \cos(\frac{pos}{10000^{2i/d_{\text{model}}}})$$

1.6.3 Multi-Head Self Attention

The transformer perceives the encoded input representation as a set of key-value pairs of dimension n.

$$\text{Attention(Q, K, V)} = \text{Softmax}(\frac{QK^T}{\sqrt{n}})V$$

With multi-head, this operation is repeated in parallel with the same Q,K,V. After what the outputs are concatenated and linearly transformed into desired dimensions.

1.6.4 ViT Block

Here we use LayerNorm which normalizes across all features for a single element in the batch.

1.6.5 Experiment and Results

For the achitecture of our model, we tested 3 models and we plotted the different accuracies and F1 scores fig. 2:

$$[\text{patches} : 7, \text{blocks} : 6, \text{hidden}_d : 8, \text{heads} : 8]$$

$$[\text{patches} : 4, \text{blocks} : 2, \text{hidden}_d : 8, \text{heads} : 2]$$

$$[\text{patches} : 7, \text{blocks} : 3, \text{hidden}_d : 16, \text{heads} : 4]$$

We found that the best results were obtained with the last one. An accuracy of 73% and a F1 score of 0.70 on the 20% validation set .

1.6.6 Discussion

The performance of our model is not very good, in terms of accuracy but also of time as it is very slow. Compared to the CNN or even MLP the transformer is not a good model for this task given it's ratio performance/speed.

1.7 PCA

We tested using PCA from 28x28 to 10x10 dimensionality reduction on MLP. Running it on the same architectures tested during MLP, we could observe that on average PCA increases slightly the f1 score and accuracy (fig. 4,fig. 3). This could mean that, both our model can capture underlying representation of our data and reduce dimensions and as well, possibly reduce the noise of our data helping thus the MLP model. As for runtime, as expected, on the [512, 256] architecture, we get 45 seconds, a 30% decrease compared without PCA, which on 75 seconds won't make a huge difference but on more complex architecture could help.

1.8 Bonus : Graded Competition

1.8.1 Data Augmentation

To try to stay on the podium during the competition we implemented data augmentation, we plotted the classes repartitions to see if the data we were given was approximately well partitioned. We realised that after training, classes 0,2,4,6 were the most misslabeled, so we looked at the dataset again. And we realized those categories were : Shirt,Coat,Pullover,T-shirt/top. So we decided to do 2 things, generate some data for classes 2 and 6, they gave us the best results and upscale the images x2 using lanczos interpolation.

We also tried to do simple data augmentation by doing some horizontal flipping, or just basic rotations. We had trouble diagnosing what was going wrong, how could teams reach 0.950 and 0.942. We managed to reach 0.946 on the validation set but then we were surprised by the outputted values of AiCrowd. We are keen to learn more next time !

2 Appendix for MS2

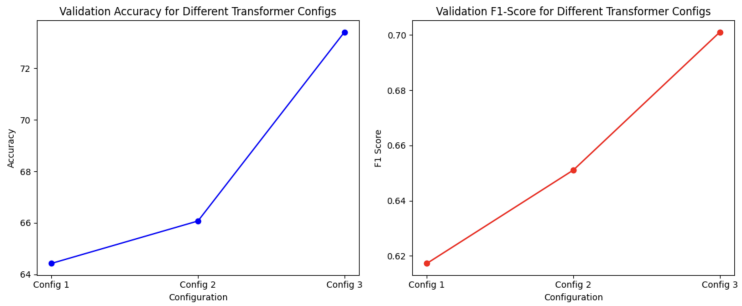

Figure 2: Performance of the Transformer model across different datasets.

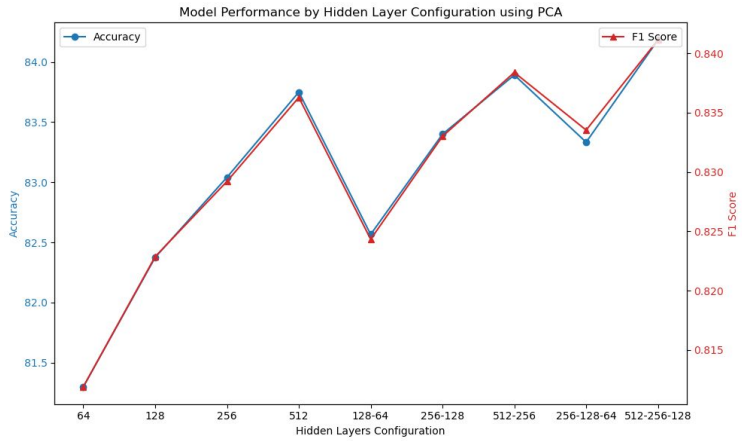

Figure 3: Comparison of model performance with and without PCA feature reduction.

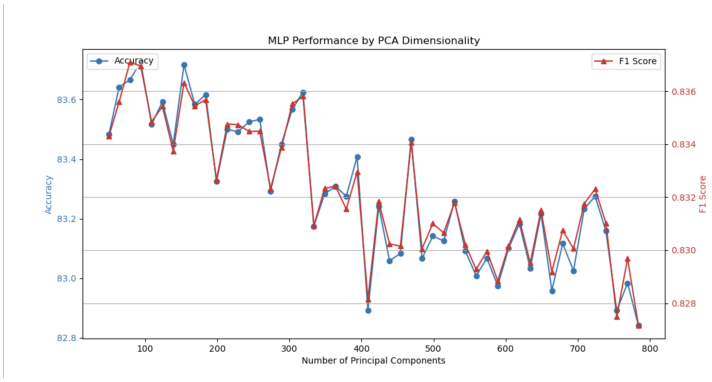

Figure 4: Comparison of model performance with different pca dimensions ranging from 50 to 784

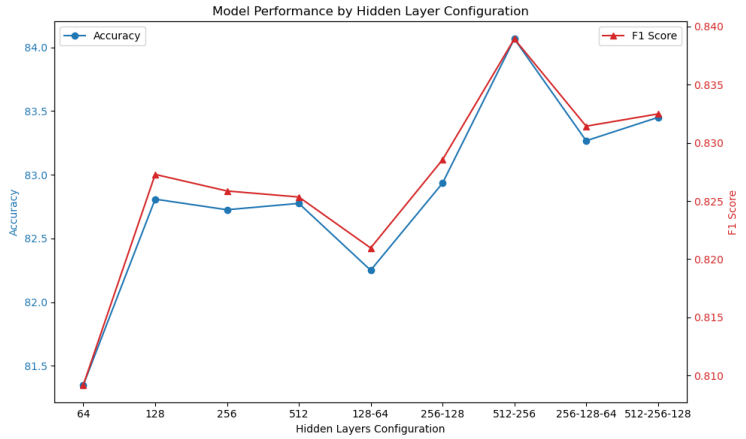

Figure 5: Overall performance metrics for the MLP model on the test set.

Layer (type)	Output Shape	Param #
Conv2d	[batch_size, 128, 28, 28]	1,280
ReLU	[batch_size, 128, 28, 28]	-
BatchNorm2d	[batch_size, 128, 28, 28]	256
Conv2d	[batch_size, 128, 28, 28]	147,584
ReLU	[batch_size, 128, 28, 28]	-
BatchNorm2d	[batch_size, 128, 28, 28]	256
MaxPool2d	[batch_size, 128, 14, 14]	-
Dropout	[batch_size, 128, 14, 14]	-
Conv2d	[batch_size, 256, 14, 14]	295,168
ReLU	[batch_size, 256, 14, 14]	-
BatchNorm2d	[batch_size, 256, 14, 14]	512
Conv2d	[batch_size, 256, 14, 14]	590,080
ReLU	[batch_size, 256, 14, 14]	-
BatchNorm2d	[batch_size, 256, 14, 14]	512
MaxPool2d	[batch_size, 256, 7, 7]	-
Dropout	[batch_size, 256, 7, 7]	-
Conv2d	[batch_size, 512, 7, 7]	1,180,160
ReLU	[batch_size, 512, 7, 7]	-
BatchNorm2d	[batch_size, 512, 7, 7]	1,024
MaxPool2d	[batch_size, 512, 3, 3]	-
Dropout	[batch_size, 512, 3, 3]	-
Flatten	[batch_size, 4608]	0
Linear	[batch_size, 512]	2,359,808
ReLU	[batch_size, 512]	-
BatchNorm1d	[batch_size, 512]	1,024
Dropout	[batch_size, 512]	-
Linear	[batch_size, 256]	131,328
ReLU	[batch_size, 256]	-
BatchNorm1d	[batch_size, 256]	512
Dropout	[batch_size, 256]	-
Linear	[batch_size, 10]	2,570

Table 4: CNN Architecture Summary

Parameter Counts

- **Total parameters:** 4,712,074
- **Trainable parameters:** 4,712,074
- **Non-trainable parameters:** 0