Advertima Fashion MNIST Assignment

First part of the assignment is about trying different CNN architectures, regularization techniques, loss functions, optimizers in order to get the best results on classifying Fashion MNIST images. For this purpose, I've implemented a few different approaches in Pytorch framework.

Data

Training data (60000 images) is split in train (50000) and validation (10000) datasets. The model with highest validation accuracy is used as a final model and this model is tested on test dataset (10000). Input image size is 28x28x1 (1 channel) and it is not zero centered (images in range (0, 1)).

Models

CNN_1

[CONV(3x3x32) -> RELU] -> [CONV(3x3x64) -> RELU] - [CONV(3x3x128) -> RELU -> POOL(2x2)] -> [CONV(3x3x128) -> RELU -> POOL(2x2)] -> [FC(3200x1024) -> RELU -> FC(1024x512) -> RELU -> FC(512x10)]

The idea is to extract features from the image using CONV and MAX POOL layers and then to make a prediction using FC nets. This initial architecture will serve as a baseline to which additional layers will be added later.
Sizes of the kernels are chosen intuitively, for CONV layer it is 3x3 with stride 1, for MAX POOL layer it is 2x2 with stride 2. Activation function is ReLU.
# of CONV parameters: (1x3x3x32 + 32) + (32x3x3x64 + 64) + (64x3x3x128 + 128) + (128x3x3x128 + 128) = 240256
# of FC parameters: (3200x1024 + 1024) + (1024x512 + 512) + (512x10 + 10)= 3807754
Proposed architecture contains ~4.000.000 parameters and training time for one epoch is ~20 s (trained on GeForce 1050Ti).

CNN_2

[CONV(3x3x32) -> **BN** -> RELU] -> [CONV(3x3x64) -> **BN** -> RELU] - [CONV(3x3x128) -> **BN** -> RELU -> POOL(2x2)] -> [CONV(3x3x128) -> **BN** -> RELU -> POOL(2x2)] -> [FC(3200,1024) -> RELU -> FC(1024,512) -> RELU -> FC(512,10)]

In this approach, batch normalization is added (Batch Norm Paper) which retains input distribution before each CONV layer. In this way it accelerates and makes training easier and more stable. Number of training epochs needed is less than the previous model. BN layer calculates and stores moving mean and variance across channels (as well as ɣ and ß learned

through backpropagation) during training, while in evaluation phase these are used for normalizing layer inputs. BN has 2 parameters (ɣ and ß) for each channel in CONV kernels. BN parameters: (32 *2 + 64*2 + 128*2 + 128 * 2) = 704

CNN_3

[CONV(3x3x32) -> BN -> RELU] -> [CONV(3x3x64) -> BN -> RELU] -> [CONV(3x3x128) -> BN -> RELU -> POOL(2x2)] -> [CONV(3x3x128) -> BN -> RELU -> POOL(2x2)] -> [FC(3200x1024) -> **DROPOUT** -> RELU -> FC(1024,512) -> **DROPOUT** -> RELU -> FC(512,10)]

Previous model is extended by adding Dropout to FC layers. Since there is a huge amount of parameters in FC layers, it may happen that model overfits. In this way, some neurons won't participate during prediction in training phase, but rather just those randomly selected with probability p = 0.5. This way model is regularized more and this could lead to better generalization on test set.

CNN_4

[CONV(5x5x32) -> BN -> RELU -> POOL(2x2)] -> [CONV(5x5x64) -> BN -> RELU -> POOL(2x2)] - [CONV(1x1x128) -> BN -> RELU -> POOL(2x2)] -> [FC(512x10)]

This model is slightly different from previous since there is only one FC layer which means very few parameters compared to *CNN_1*, *CNN_2*, *CNN_3*. Moreover, first two CONV layers have kernels of size 5x5 which capture bigger spatial space in input image, while the third CONV has kernel 1x1 and its function is to increase number of channels and to make embedding from the previous layer.
Total number of parameters for this model is ~ 65000 and one epoch training time is ~

CNN_5

[CONV (3x3x32) -> BN -> RELU] -> [CONV(3x3x64) -> BN -> RELU] - [CONV(3x3x128) -> BN -> RELU -> POOL(2x2)] -> [CONV(5x5x128) -> BN -> RELU] -> [CONV(5x5x256) -> BN -> RELU -> POOL(2x2)] -> [FC(1024x512) -> RELU -> FC(512x10)]

This model has 5 CONV layers with different kernel sizes, Batch Norm after each of them, 2 MAX POOL layers that encapsulate spatial information and make it more robust on changes. At the end, there are 2 FC layers that make predictions based on extracted features. Number of parameters is ~1800000, and one epoch training time is ~32s. Since the most computationally intensive operation is convolution, adding extra CONV layers increases training time even though number of parameters is not that much increased.

Approach

For all models training is executed with and without augmentation and with two different types of final layers (Softmax and Sigmoid). Adam optimizer is to optimize cross entropy loss with fixed learning rate and weight decay which enables L2 regularization loss.
Augmentation is done using *random crop* and *random horizontal split*. This will enrich the dataset for training and allow better generalization on test set (expected behavior). Also, it is expected that training lasts longer then without augmentation since data is auxiliary bigger and validation set can not be overfitted easily.

For the comparison purposes, all the models are trained with same hyperparameters (batch_size=128, learning_rate=4e-3, L2 weight decay = 1e-3, dropout probability = 0.5, number of epochs = 10). This is done in script ***train.py*** by running this command:
python train.py --model CNN_1 --batch_size 128 --lr 0.004 --rs 0.001 --num_epochs 10 --augment_data True --loss_type softmax

Table 1 presents the resulting test accuracy for different models.

Model	No Augment. Softmax	Augment. Softmax	No Augment. Sigmoid	Augment. Sigmoid
CNN_1	89.16%	89.13%	84.11%	83.38%
CNN_2	**90.25%**	**89.43%**	85.85%	86.61%
CNN_3	88.50%	88.37%	81.56%	84.72%
CNN_4	86.95%	85.59%	81.20%	79.39%
CNN_5	88.23%	88.01%	85.93%	84.98%

Table 1: Test accuracies for different approaches

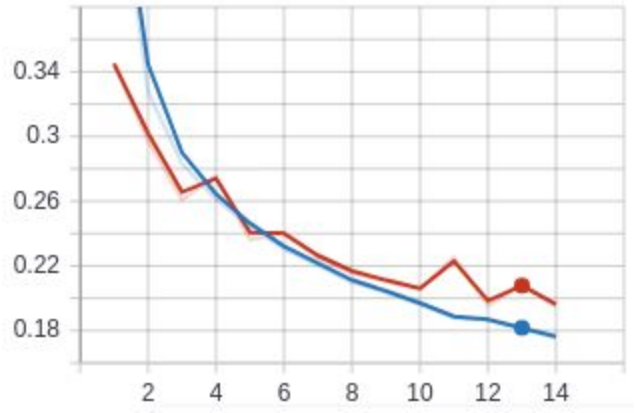
It turns out that model *CNN_2,* which uses Batch Normalization, performs the best across most approaches. Also, Softmax outperforms Sigmoid in all cases (maybe because it considers all classes in its probability estimation). Data augmentation can be used and may lead to solid results, but number of training epochs should be increased then. *CNN_3* with Dropout performs worse than *CNN_2*, this may come from the fact that Batch Norm reduces the need for Dropout since it already does some kind of regularization. Model *CNN_4* lacks capacity, but it results in solid accuracy regarding its complexity and size. Considering *CNN_5* complexity it was expected to have the best accuracy, however this is not the case.

Specific model analysis

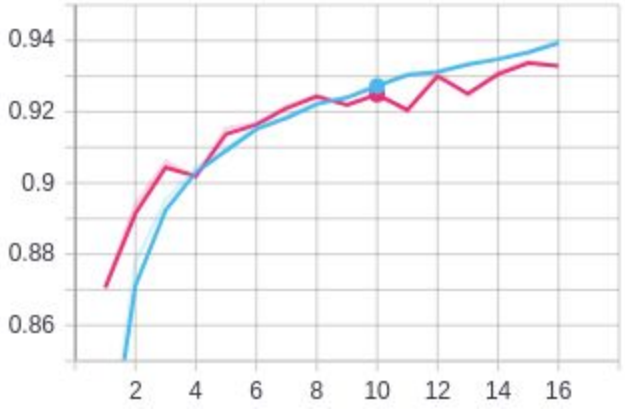
Model **CNN_2 with Softmax and Data Augmentation** will now be further analyzed.

Finding optimal hyperparameters (initial learning rate, L2 regularization strength, batch size, num_epochs) for the specified model is conducted in ***hyperparameters_search.py*** script.

Optimal number of epochs (10) is decided based on the train and val loss function across epochs in Picture 2 and Picture 3. After 10 epochs validation loss / accuracy stops to decrease / increase significantly (this is the moment where overfitting starts).



Picture 2: Train and val loss (blue, red)



Picture 3: Train and val accuracy (blue, red)

Optimal parameters:
Number of epochs = 10
learning_rate = 0.0003593
batch_size = 64
num_epochs = 10
L2 reg_strength = 0.0003593

Test accuracy for these parameters is **92.91%**.

Conclusion

Simple stacking convolutional and pooling layers from some point does not get significant improvement in accuracy (shown from example *CNN_4*, *CNN_2*, *CNN_5*). Moreover, it may downgrade performances. Therefore, new types of architectures must be used.
Further improvements in accuracy could be found in novel architectures (residual, inception, dense blocks) which enables better feature extraction and signal (gradient) propagation.