Introduction to Machine Learning (SS 2024)
Programming Project

Author 1
Last name: Drexel
First name: David
Matrikel Nr.: 01417923

Author 2
Last name: Drexel
First name: David
Matrikel Nr.: 01417923

I. Introduction

For this report we consider the task of classifying images in the MNIST dataset. MNIST contains a total of 70000 instances of handwritten digits in a 28×28 pixel grayscale format. Accordingly, this is a multilabel classification task, with each image corresponding to a digit from 0 to 9.

The dataset is fairly balanced overall, with the frequency for each digit ranging from 9.01% to 11.25%. There is a predefined split into a 60000 sample training set, and a 10000 sample test set which we keep as-is.

II. Implementation / ML Process

We decided to implement a convolutional neural network (CNN), as well as a multilabel support-vector-machine (SVM) classifier for this task.

A. Data preprocessing

In order to perform unbiased hyperparameter search, we split off 10000 images of the original training set to use as a validation set, leaving us with a reduced training set size of 50000.

For the CNN classifier we symmetrically pad each image with zero values from 28×28 to 32×32, following the paper which introduced MNIST and LeNet-5 [1]. This allows the center of the convolution in the first layer to sweep over all features of each digit, which is advantageous for learning. We further apply whitening with mean 0.1307 and standard deviation 0.3081, which correspond to the statistical properties of the original training set scaled by a factor $\frac{1}{255}$.

For the SVM classifier we simply scale by $\frac{1}{255}$ to obtain features in the range $[0,1]$. We found this to be necessary in order for the computation to converge in practice.

B. Classifier architecture

The CNN classifier is based on the LeNet-5 architecture and is implemented using PyTorch. Figure 1 shows the neural network architecture. We chose this method because neural networks with convolutional layers are a natural fit for classification tasks involving images. Convolutional layers explicitly take into account the spatial relationship of neighbouring pixels, which is naturally quite important for interpreting images. Due to their re-use of kernel weights across the entire input, they further have a form of translational pseudo-invariance, which can help generalization.

The SVM classifier consists of an ensemble of SVMs and is implemented using scikit-learn. Since a vanilla SVM can only be used for binary classification, we fit an ensemble of 10 binary classifiers, one for each of the classes. The predicted class label for any given instance is then dictated by the ensemble member which predicts membership with the highest confidence. We chose this method because we never used SVMs for multilabel classification and were curios about their performance. They are suitable for this classification task, as long as the classes are either linearly separable, or we can find a kernel such that they are separable in the resulting implicit feature space. Naively we'd expect SVMs to perform worse than CNNs as they are not making use of the spatial relationship between pixel features, and the classification task is likely to be highly non-linear.

We decided against using the k-nearest neighbors algorithm for this task due to the high-dimensional nature of the image data, which would have likely lead to poor performance. Decision trees would likely also perform poorly on the raw images themselves, as individual pixels are relatively uninformative features. We hypothesize however, that both these methods could obtain good results when combined with some form of dimensionality reduction.

C. Hyperparameters

We implement simple random search for our hyperparameter optimization. For each hyperparameter we define either a numerical range $[a,b]$, or a discrete set of choices $\{c,d,e,\dots\}$. Over the course of 60 trials per classifier we uniformly sample hyperparameter configurations according to these bounds, and train a model on the training set data. We select the best set of hyperparameters based on the resulting models performance on the validation set.

For the CNN classifier we train for a fixed 10 epochs using `AdamW`. Other than the learning rate hyperparameter, we use PyTorchs default parameters for the optimizer. We optimize across batch size B, learning rate l_R, hidden layer sizes n_1, n_2, and choice of activation function. The search bounds, as well as our final hyperparameters, are shown in Table I.

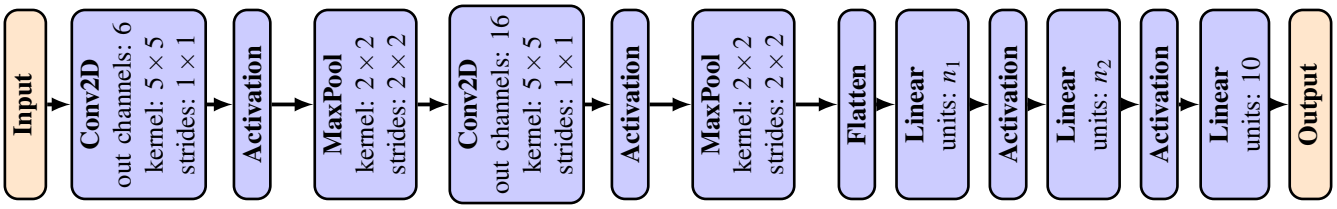

Fig. 1. Architecture of the CNN classifier. Hidden unit sizes n_1, n_2 are optimized via hyperparameter search.

	B	l_R	n_1	n_2	Activation
Bounds	$[8, 128]$	$[0.0001, 0.1]$	$[32, 256]$	$[32, n_1]$	{ReLU, Sigmoid}
Final choice	88	0.0078627	68	61	Sigmoid
	x	y	Kernel		
Bounds	$[-2., 3.]$	$[-4., 0.]$	{Linear, Poly, RBF, Sigmoid}		
Final choice	1.2176867	-1.6679861	RBF		

TABLE I

HYPERPARAMETER SELECTION FOR THE CNN (TOP) AND SVM (BOTTOM) CLASSIFIER.

	Train set	Validation set
CNN	99.592%	98.69%
SVM	100%	98.49%

TABLE II

FINAL CLASSIFIER ACCURACY ON TRAINING AND VALIDATION.

For the SVM classifier we use scikit-learns inbuilt support vector classifier `sklearn.svm.SVC`. We use the default parameters, except for the choice of kernel function, the regularization parameter C, and the kernel coefficient γ. Instead of drawing from a uniform distribution for C and γ, we uniformly draw exponents x and y such that $C = 10^x$ and $\gamma = 10^y$. This is done to achieve a better distribution in the search space, which is usually stepped through in powers of 10 when manually choosing values. The search bounds for these hyperparameters, as well as our final choices are again depicted in Table I.

Due to time constraints, we randomly sample 10000 instances from the training set each trial, and only fit the SVM classifier to this subset instead of the entire training set for the hyperparameter search. This is done to significantly lessen the computational effort of the search, which scales quadratically with the number of training instances. We then refit with the optimized hyperparameters on the entire training data to determine the final validation and test set performance.

III. RESULTS

Both classifiers achieve good results on the validation set. The obtained accuracies are shown in Table II.

IV. DISCUSSION

The random hyperparameter search ended up outperforming the values we initially hand-picked from assumed best practices. It did however lead to quite a lot of wasted computational effort, due to runs which were hamstrung by non-viable parameter choices from the get go. In particular, for the CNN classifier we observe 47 out of 60 training runs not converging from the very beginning of training. This could likely be mitigated by imposing tighter bounds on the parameter search. Another idea would be to implement early stopping to abort runs which show no convergence after only a few epochs.

We initially attempted to train the SVM classifier on the raw pixel data without any normalization. This however lead to apparent convergence issues and we had to manually interrupt the training due to time constraints. It appears the normalization confers significant numerical benefits to the underlying optimization problem.

Further improvements could likely be achieved by performing data augmentation.

V. CONCLUSION

We achieve a final test set performance of 98.69% with the CNN classifier, and 98.49% with the SVM classifier. To us, it is somewhat surprising how competitive the accuracy of the SVM classifier is compared to the CNN. Despite not being naturally suited for image data, SVMs turn out to be a valid choice of method for simple multilabel image classification. We however expect this balance to more heavily tip in favor of neural networks when scaling to larger amount of classes, or more training data.

These results also highlight the utility of using random search for hyperparameter selection. It appears somewhat counter intuitive how this method can compete with e.g. a principled grid search, but it manages to obtain good results with only few trials. It therefore offers an extremely simple alternative to more involved methods like Bayesian Optimization or Hyperband [2].

REFERENCES

[1] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[2] Lisha Li et al. “Hyperband: A novel bandit-based approach to hyperparameter optimization”. In: *The Journal of Machine Learning Research* 18.1 (2017), pp. 6765–6816.