Recruitment task

Image classification

Agata Skibińska

30.04.2021

Takeaways are described in section 6.

1 Introduction

The goal of this task is to classify images from delivered dataset into one of 36 classes. To achieve this goal, two approaches were compared - a classical classification model and a deep learning model. The chosen types were Support Vector Machine and Convolutional Neural Network accordingly. The motivation for using two models is that deep learning method require much more memory and time to train. Using smaller model can give a baseline of what results can be achieved without testing various architectures or hyperparameter tuning. However, the focus was on CNN model. Several experiments were conducted to find the best architecture. Lastly, both methods were analysed concerning errors they make.

2 Data analysis

The delivered dataset consists of 30134 examples of handwritten letters and digits with a label from one of 36 classes. The images were probably delivered from MNIST dataset. Some examples of images are presented in figure 5. The labels for digits do not correspond with the digit in the image.

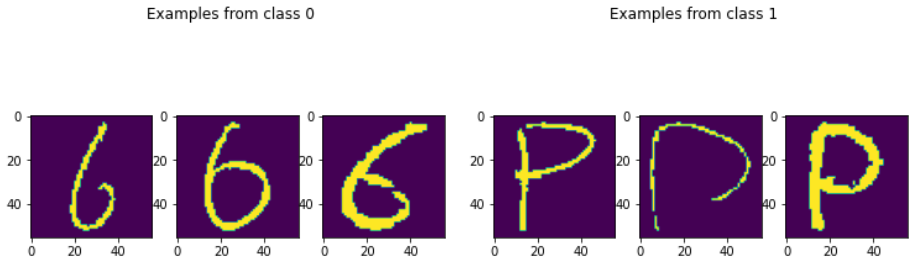

Figure (1) Examples from dataset

The dataset is not balanced and the histogram of labels is illustrated by figure 2. The class with the smallest size is 30 with only one example. The biggest class is 14 with 3291 examples. This problem needs to be addressed in order to achieve good results.

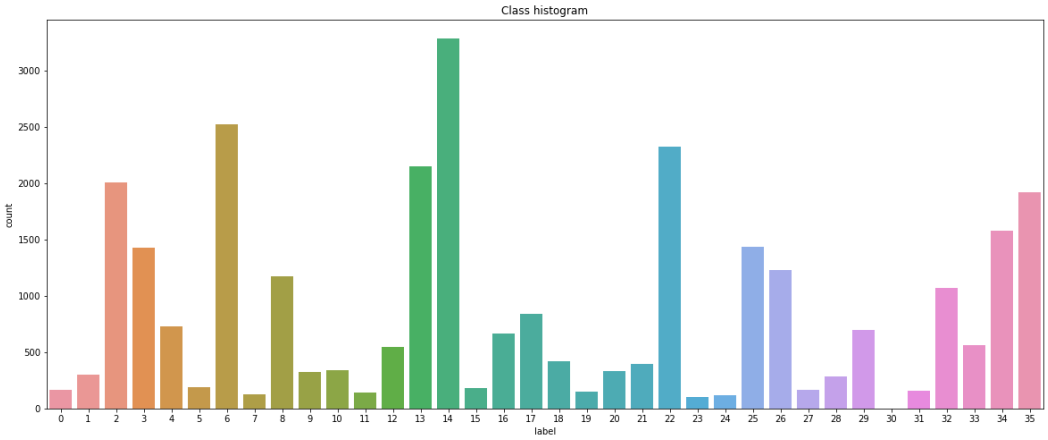

Figure (2) Class histogram

3 SVM

3.1 Implementation

The support vector machine classifier was implemented using sklearn library. All hyperparameters are using default values, since it was only used to compare results. The motivation for using SVM is that it is a high-quality model that gives good results.

3.2 Evaluation

The classifier was evaluated using stratified cross validation with 5 folds. One sample from class with label 30 was manually moved to training set. The results of training are presented in figure 3. The numeric values are presented in table 1.

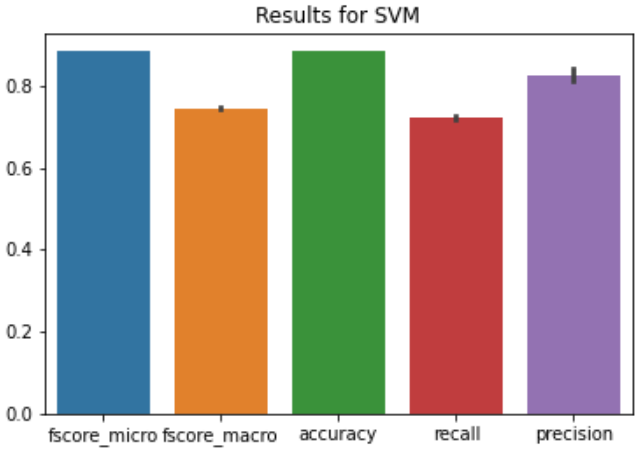

Figure (3) SVM results

metric	value
fscore_micro	0.884047
fscore_macro	0.745882
accuracy	0.884047
recall	0.723854
precision	0.826962

Table (1) SVM results

3.3 Error Analysis

The confusion matrix for SVM is presented in figure 4. The highest number of errors between classes is between class 10 and 2 that correspond to zeros and letter O. Some of examples are shown in fig 5. This was expected because it is hard to distinguish these images even by human. Examples from class with smaller support in training set are classified into similar class with higher support. Similarly, the model confuses true class 9 and predicted 25. The class of 9 stands for 1 and 25 for letter i. However, the size of class 9 is much smaller than class 25, so the model predicts the latter class more frequently.

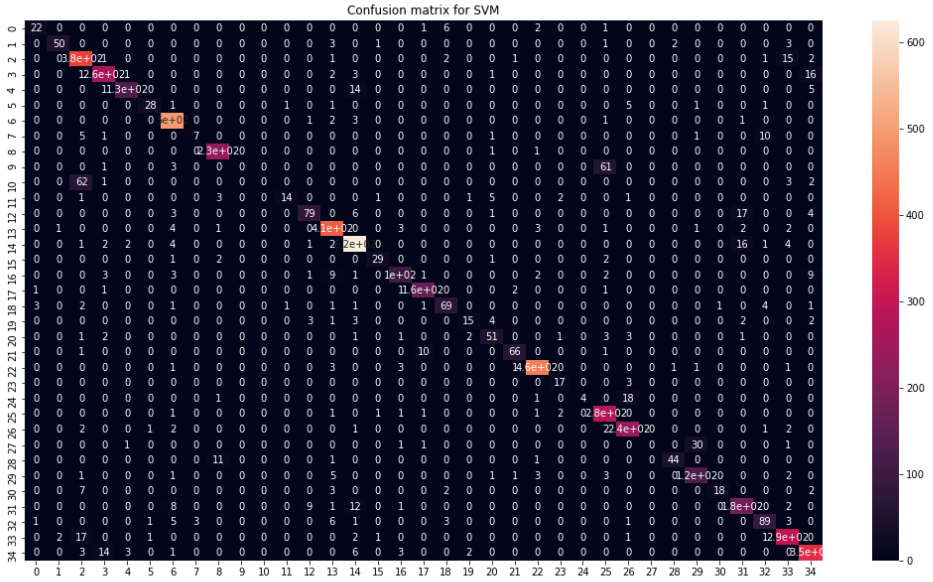

Figure (4) SVM confusion matrix

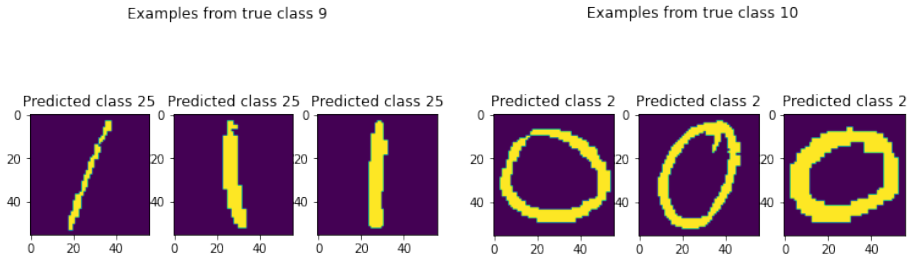

Figure (5) Examples of wrongly classified

4 CNN

4.1 Implementation

All deep models were implemented using Tensorflow 2.4.1.

4.1.1 Model selection

Train and test set were created using train_test_split function with test set size of 0.33. One sample from class with label 30 was manually moved to training set. Because of high GPU and memory usage, cross validation was not used to evaluate the models.

4.1.2 Preprocessing

The provided dataset was already normalized and binarized. The only preprocessing used was the reduction of the size of the images four times, from the original size 56x56 to 28x28. The reduced images did not lose their

interpretability. This change allowed to significantly reduce the size of trained models. An example of image before and after size reduction is presented in figure 6.

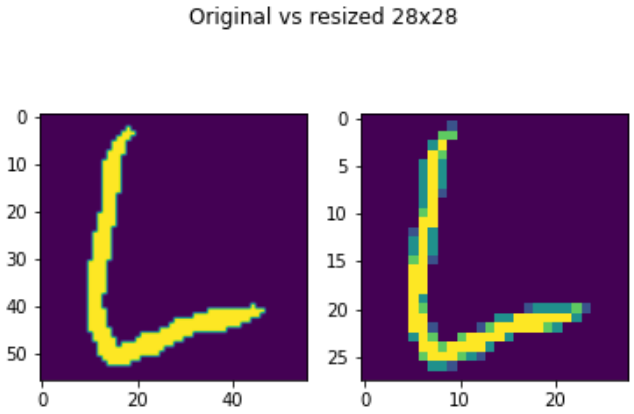

Figure (6) Example of resized image compared to original

4.1.3 Layers

In the next sections, there is more detailed description of tested models. However, the default building blocks are:

- Convolution layer - Conv2D layer with default stride (1,1), no padding, using bias and default kernel initializer glorot_uniform;
- Pooling layer - MaxPool2D layer with default pool size (2,2), no padding and no stride.

4.1.4 Hyperparameters

All models in the next sections except the last were trained during 20 epochs (35 epochs in last) using Adam optimizer and mini batch of size 80. Learning rate was annealed during training with 5% pp change every epoch.

The used loss function is categorical crossentropy.

4.2 Experiments

In order to find the optimal CNN architecture, following questions need to be answered:

1. How deep should the model be?
2. How many filter maps should the Conv2D layers have?
3. What is the best size of dense layer?
4. Which regularization technique works the best and/or which percentage of dropout gives the best results?
5. Is the training dataset enough or is there a need for data augumentation?

4.2.1 Model depth

This experiment aimed to determine how many convolution - subsampling pairs should the model have. The combinations of tested layers follow:

- Input - Conv2D(filters=24, kernel_size=5) - MaxPool2D - Dense(256) - Dense(10)
- Input - Conv2D(filters=24, kernel_size=5) - MaxPool2D - Conv2D(filters=48, kernel_size=5) - MaxPool2D - Dense(256) - Dense(10)
- Input - Conv2D(filters=24, kernel_size=5) - MaxPool2D - Conv2D(filters=48, kernel_size=5) - MaxPool2D - Conv2D(filters=64, kernel_size=5) - MaxPool2D - Dense(256) - Dense(10)

After third MaxPool2d layer the image would be reduced to 4x4 (with padding *same*). So it makes no sense to add fourth pair.

The results of these models are presented in figure 7. The worst results were achieved using only one pair [Conv2d, MaxPool2d]. The results of 2 and 3 pairs are very similar, but for computational cost the architecture with 2 pairs will be used.

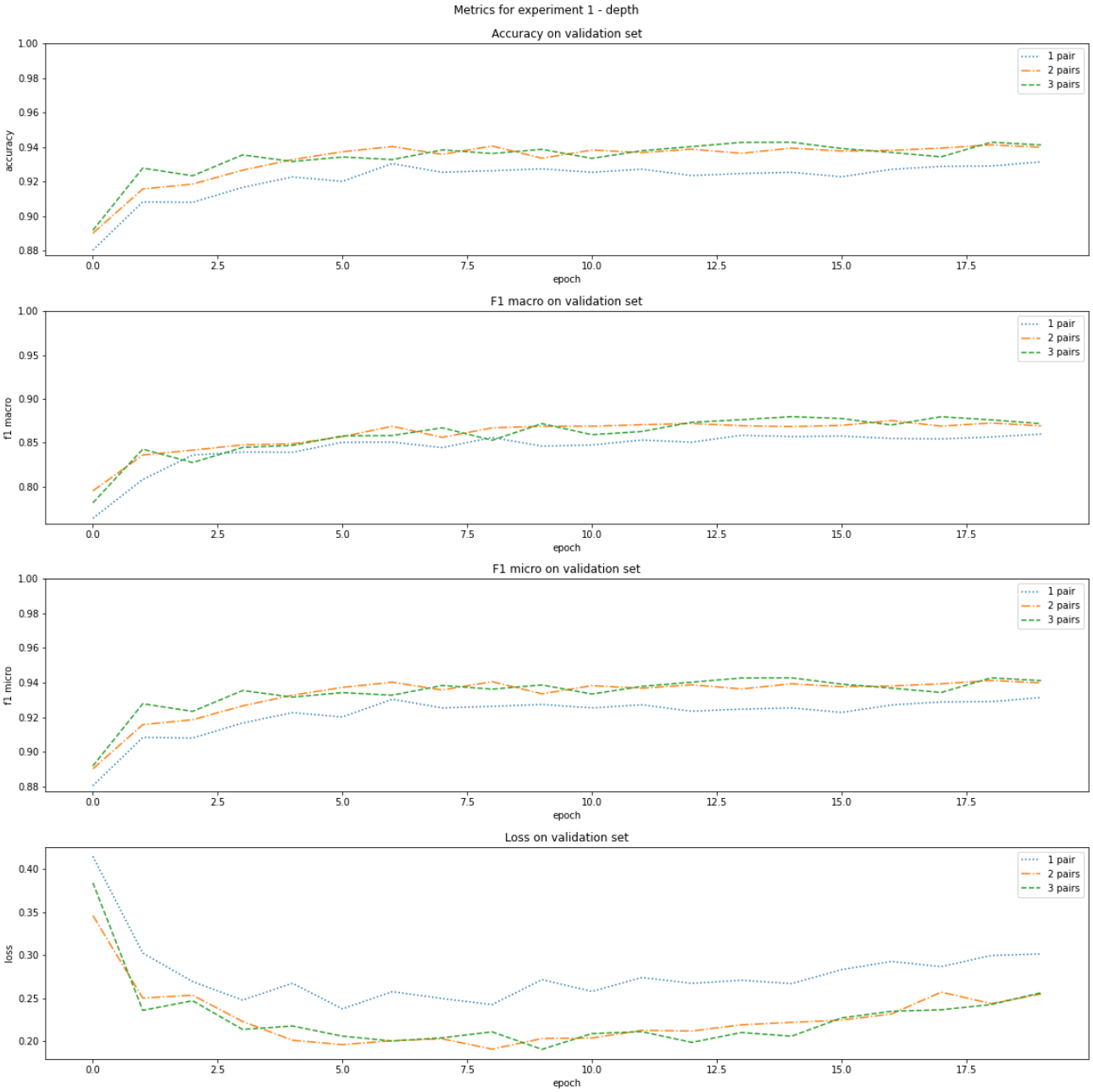

Figure (7) Results of first experiment - Model depth

4.2.2 Feature map size

This experiment was expected to determine, which combination of feature map sizes will be used. The tested combinations follow (for clarity, Input and Dense layers will be omitted):

- Conv2D(f=8, k_s=5) - MP2D - Conv2D(f=16, k_s=5) - MP2D
- Conv2D(f=16, k_s=5) - MP2D - Conv2D(f=32, k_s=5) - MP2D
- Conv2D(f=24, k_s=5) - MP2D - Conv2D(f=48, k_s=5) - MP2D
- Conv2D(f=32, k_s=5) - MP2D - Conv2D(f=64, k_s=5) - MP2D
- Conv2D(f=48, k_s=5) - MP2D - Conv2D(f=96, k_s=5) - MP2D

The results are shown in figure 8. The best filter sizes are 32-64 and 48-96, but the difference is very small and not worth the computational cost of achitecture with more feature maps.

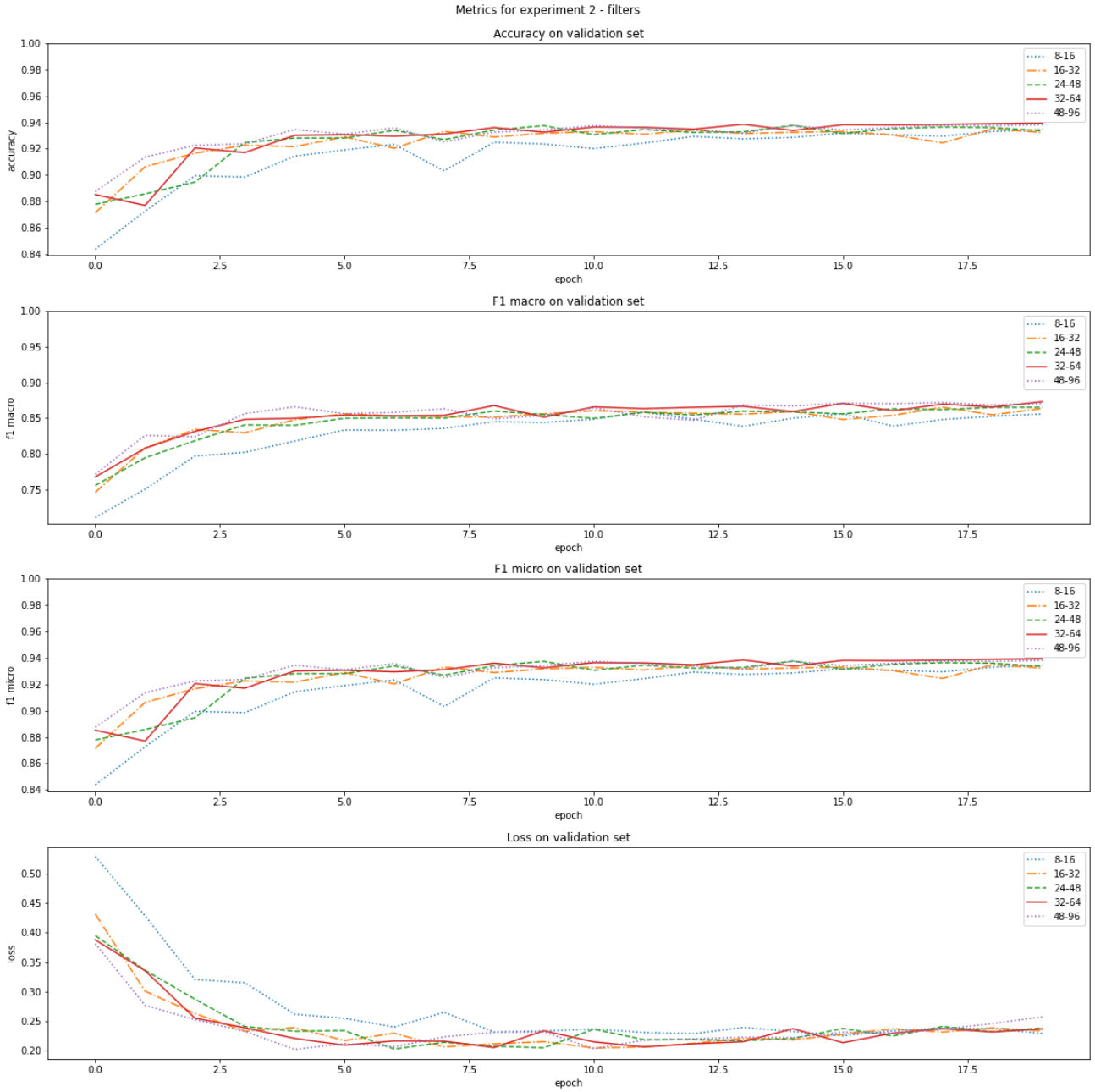

Figure (8) Results of second experiment - Filter size

4.2.3 MLP layer size

This experiment tested several suggested fully connected layer sizes:

- 0
- 32
- 64
- 128
- 256
- 512
- 1024

The results are shown in figure 9. Dense layers with more parameters perform slightly better. Again, taking into consideration the computational cost and memory usage, the best choice seems to be 256. This size was used in further experiments.

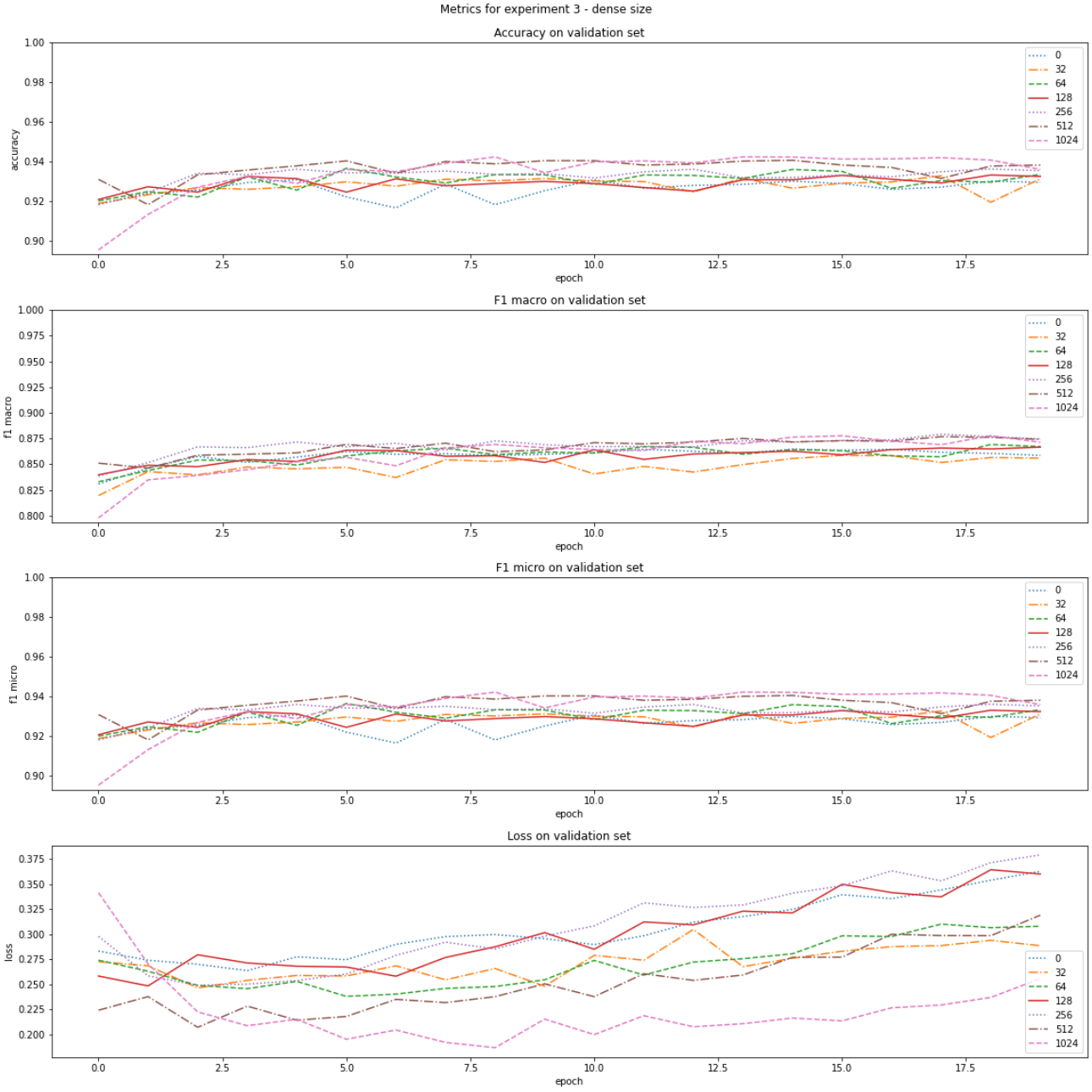

Figure (9) Results of third experiment - Fully connected layer size

4.2.4 Dropout percentage

One of widely used regularization techniques is dropout - it prevents the network from overfitting by "turning off" random units. The amount of units dropped is specified by percentage. The tested values of dropout are [0%, 10%, 20%, 30%, 40%, 50%, 60%].

The results are shown in figure 10. The best dropout percentage is 30%. Using dropout resulted in smoothing the loss curve - this indicates that the models is learning better.

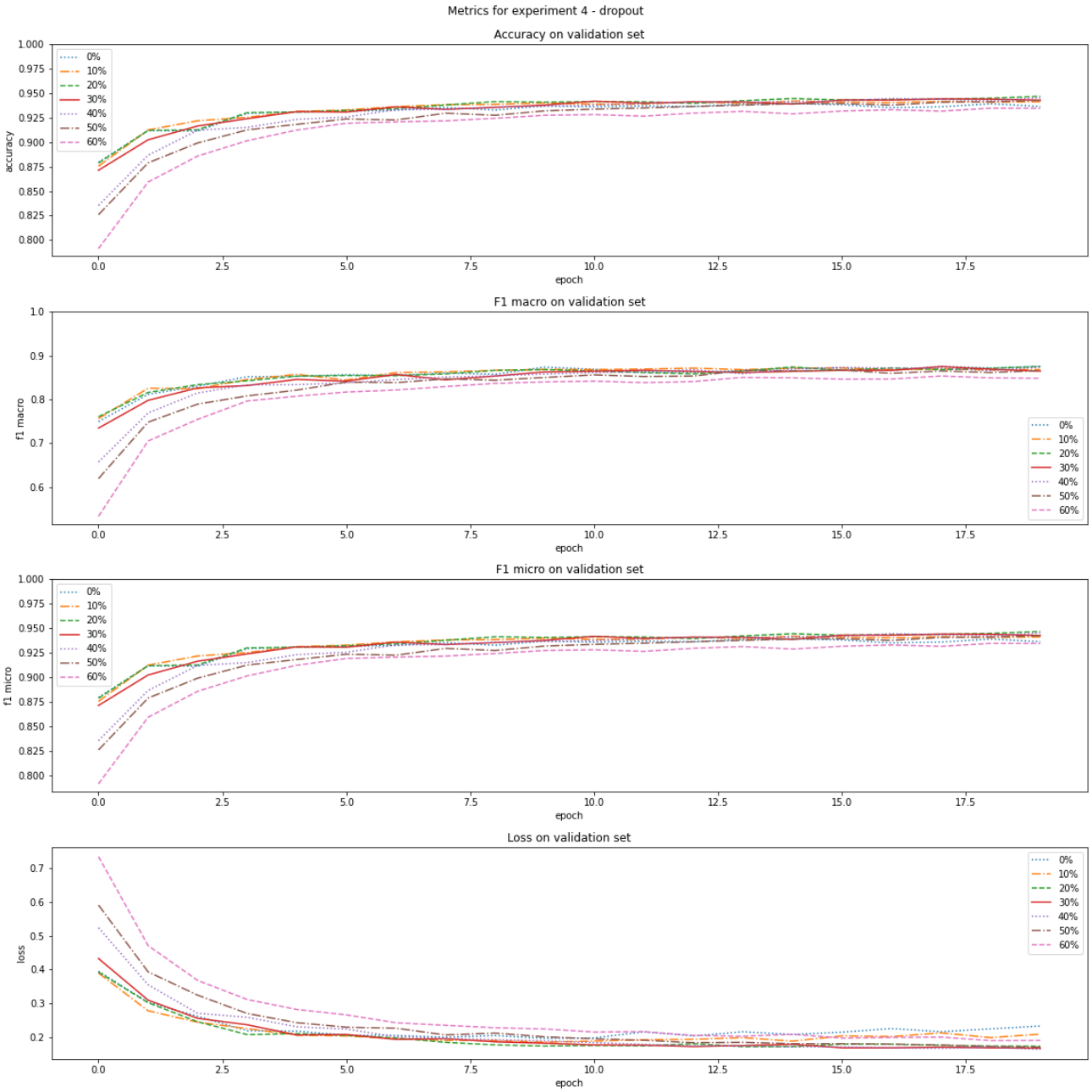

Figure (10) Results of fourth experiment - dropout percentage

4.2.5 Additional regularization

Lastly, this experiment compared different additional techniques added on top of model built in previous experiments. Model combination testes follow:

- + Early Stopping - although it's only to see if the model needs less epochs than specified
- + Batch Normalization added between Conv2D and Activation Layer
- + Data Augumentation - used to create traning dataset (10 degree random rotation range, 0.1 random zoom range, 0.1 width and height shift range, binarized to mimic original data)
- MaxPool replaced by Conv2d with stride (2, 2) (P ->C abbreviated) - this gives the architecture more nonlinearity and makes this layer learnable
- P ->C + Data Augumentation

The results are shown in figure 11. Early Stopping was not used by any of the tested models. The best results are achieved using model P ->C+DA which stands for model with MaxPool replaced by Conv2d with stride (2). This model achieves the best results in all metrics.

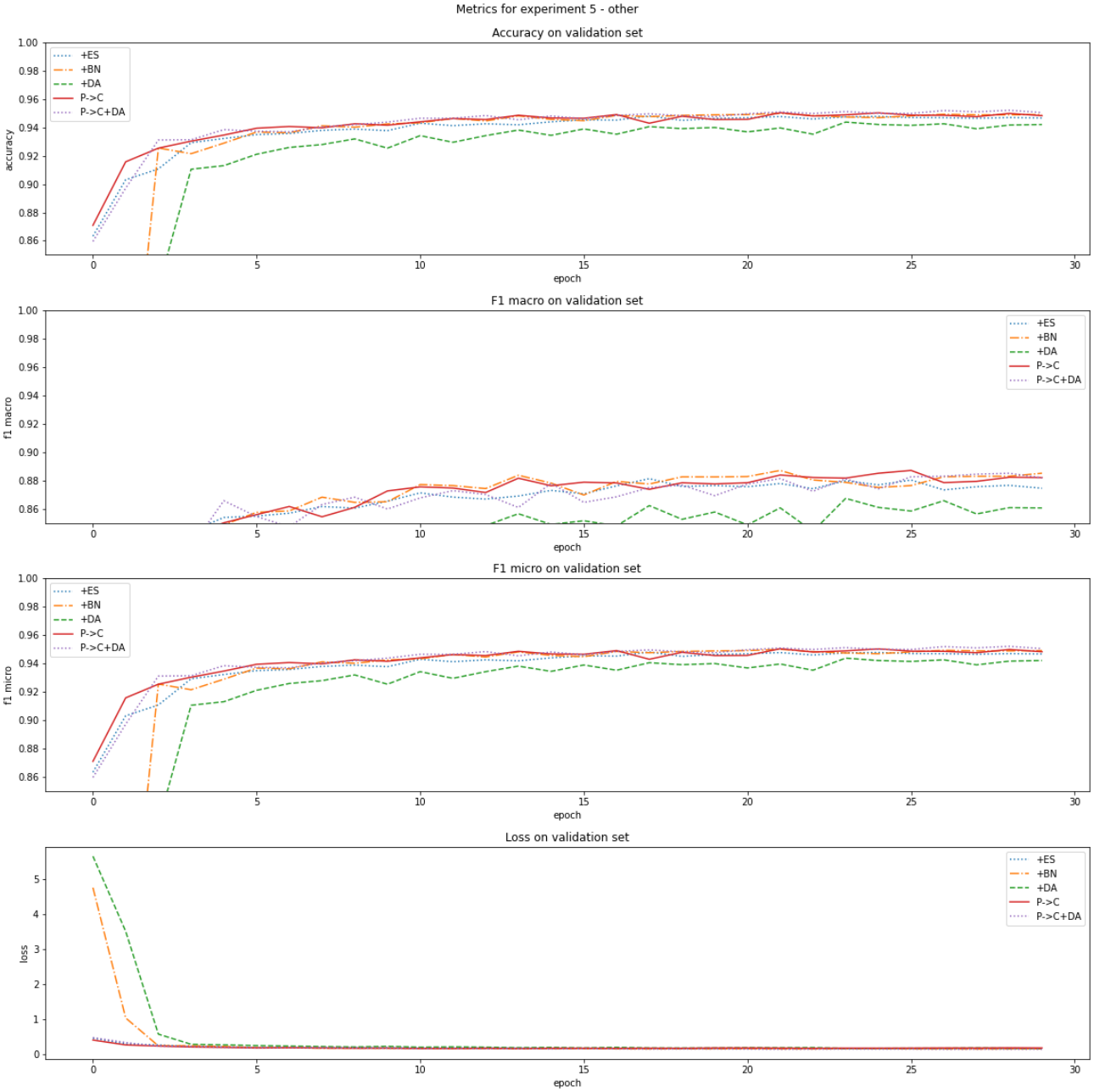

Figure (11) Results of fifth experiment - regularization techniques

4.2.6 Class weights regularization

One of techniques of dealing with class imbalance offered in Keras models is using class weights. This adds 'more attention" for under-represented classes by weighting the loss function. The results are shown in figure 12. Unfortunately this technique did not work well, although the loss decreased faster.

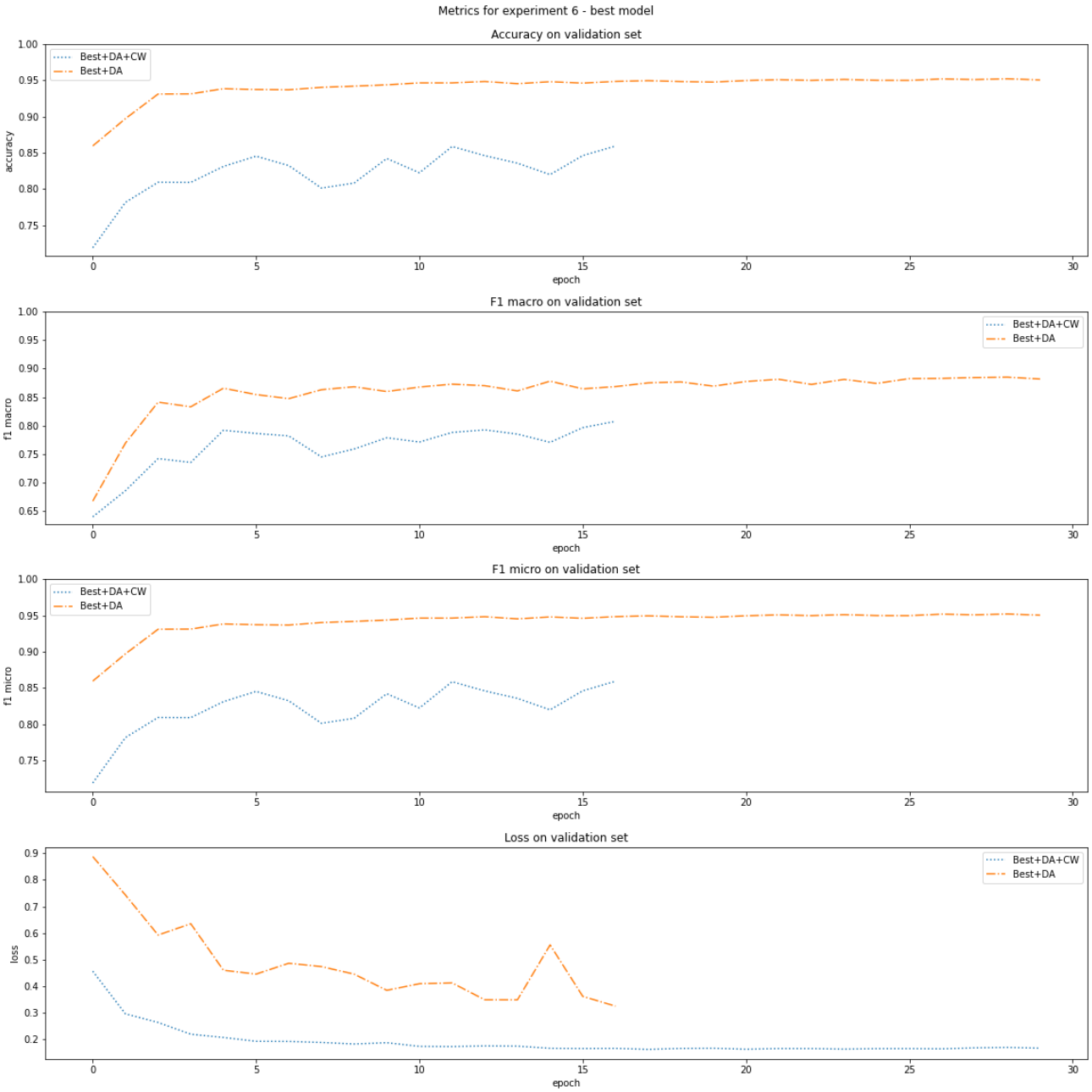

Figure (12) Results of adding class weights regularization

4.3 CNN vs SVM

The results of the best model are shown in table 2. Using CNN architecture allowed to gain significant improvement in all metrics.

metric	SVM	CNN
fscore_micro	0.884047	0.952137
fscore_macro	0.745882	0.885097
accuracy	0.884047	0.952136

Table (2) SVM results

4.4 Error Analysis

The confusion matrix for CNN best architecture model is presented in figure 13. In comparison to SVM, the CNN model share the problem of confusing similar digits and letters: O with 0 and I with 1, but makes less errors besides that. This indicates that the model works well and the only problem is the size of training classes.

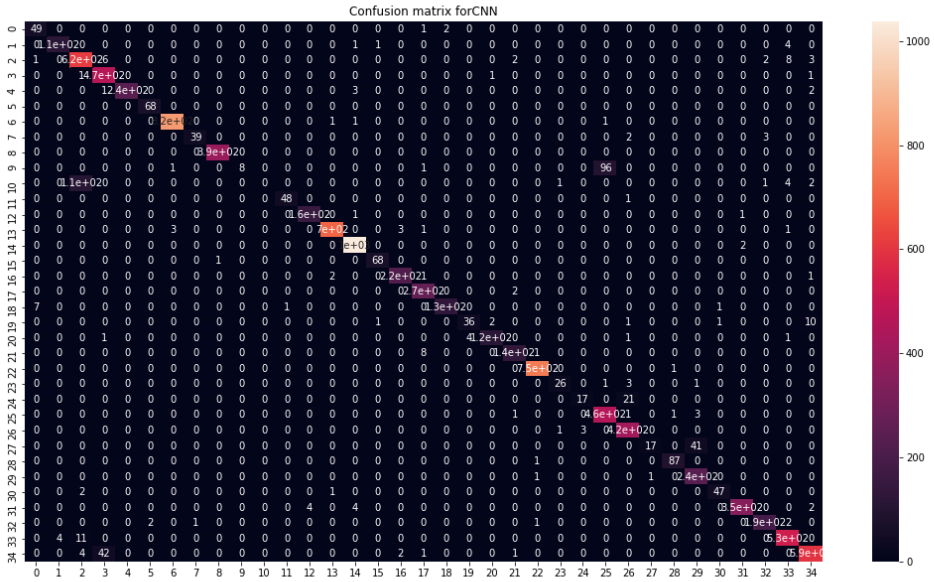

Figure (13) CNN confusion matrix

5 Conclusions

5.1 Hyperparameters

Despite the experiments described in previous sections, there are some hyperparameters that were not tested, e.g the optimizer, the loss function, augumentation techniques, weight initializers. Testing other alternatives could deliver interesting insights.

5.2 Imbalanced dataset

The first problem with this dataset is its imbalance. Since this dataset is a filtered version of original MNIST (digits) and EMNIST (letters) datasets, one of the possible solutions would be to use the rest of the samples from these sources. Alternatively, finding another source of training data would be a great addon. Another solution would be to use some kind of data-generating model to generate samples from undersampled classes. Additional data would make great testing set as well.

5.3 Better model

CNNs are great in dealing with image data, but there are newer and fancier architectures that could potentially give better results. Using models with pretrained layers like Resnet could give better scores. An interesting approach would be to use some kind of context data (e.g words the letter/digit combinations make) to determine the probabilities of a given letter/digit. This would have the potential to solve the problem of confusing digits with letters (digit 1 with letter f).

6 Takeaways

1. The provided dataset is imabalanced and this significantly affects the classification results. Using additional data from undersampled classes would make a great improvement.

2. Using SVM gave good results (fscore macro 0.75) but deeper approach allowed to score 0.88 in the same metric.

3. Recommendation for next step would be to use transfer learning and models like Resnet with pretrained layers.