Computer Science 6915 - Winter 2019, Assignment 4, Kaushlender Kumar and Bhumi Tailor

Git Hub web link:

https://github.com/kaushlenderk/CNN_SVHN

1. CNN model architecture

The CNN model to classify the house numbers consists of the following layers:

1. **First, convolutional input layer:**
   Layer with 32 units, input shape (32, 32, 3) and RELU as an activation function. Batch normalization to normalize first layer output data.

2. **Second, convolutional hidden layer:**
   Layer with 32 units and RELU as an activation function. Applied batch normalization, 2D max polling of (2,2) and dropout of 0.25 as additional layers.

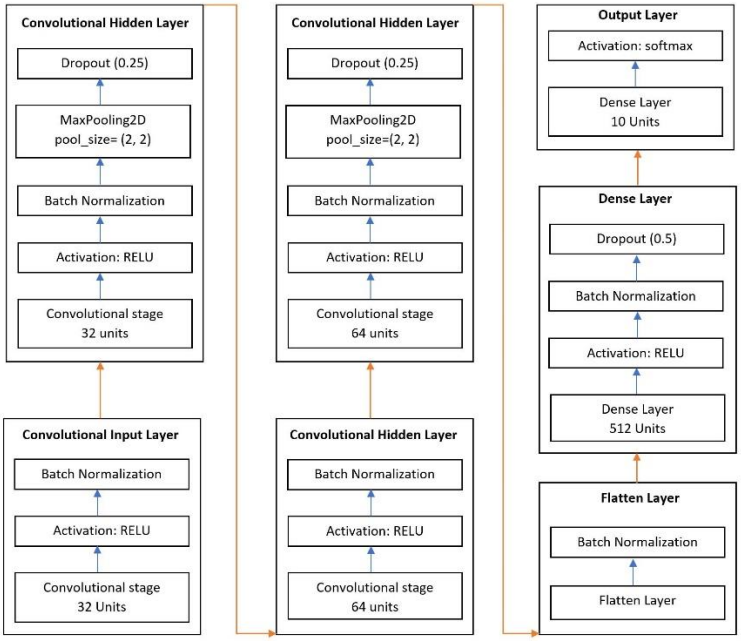
3. **Third, convolutional hidden layer:**
   Layer with 64 units and RELU as an activation function and applied batch normalization.

4. **Fourth, convolutional hidden layer:**
   Layer with 64 units and RELU as an activation function. Applied batch normalization, 2D max polling of (2,2) and dropout of 0.25 as additional layers.

5. **Flatten layer:**
   Added flatten layer with batch normalization to transform data before feeding to the dense Layer.

6. **First dense layer:**
   Added dense layer with 512 units and RELU as an activation function. Applied batch normalization and dropout of 0.5 as additional layers.

7. **Second dense layer (Output Layer):**
   Added output layer as a dense layer with 10 units and softmax as an activation function to predict house number.



2. Reason to select this architecture

We selected the CNN architecture to train our final model based on the obtained accuracy and loss (categorical crossentropy) function performance on the validation dataset over other architectures. We tried to train our model with or without batch normalization and 2D max pooling layers on previous layers output, and we found a consistent increase in its performance on each epoch run with these two layers, better than without them. Also, the addition of the Dropout layers and another 64 units convolutional hidden layer showed an increase in the model performance. The Dense layer with 512 units showed a little bit better performance compared to the one with 128 units.

3. How the network was trained

We decided to use "extra_32x32" as training dataset to train our model because we observed a much better performance from the first epoch run (starting accuracy in range of 90 percent) compared to the performance on "train_32x32" dataset. We also added "EarlyStopping" as callback function with patience value of 5 to monitor the validation data "loss" performance and to stop the process in case of no further improvement to prevent the model from the overfitting.

Model learning process setting:

Loss function: categorical_crossentropy
Optimizer: SGD as optimizer function "SGD (lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)"
Metrics: measure model accuracy based on "accuracy" matrix
Epoch: 20

4. Preprocessing

We performed normalizations of the test and train datasets by scaling the value in a range between 0 and 1. This was done by dividing the value by 255 before feeding it to the CNN model.

5. Performance matrix

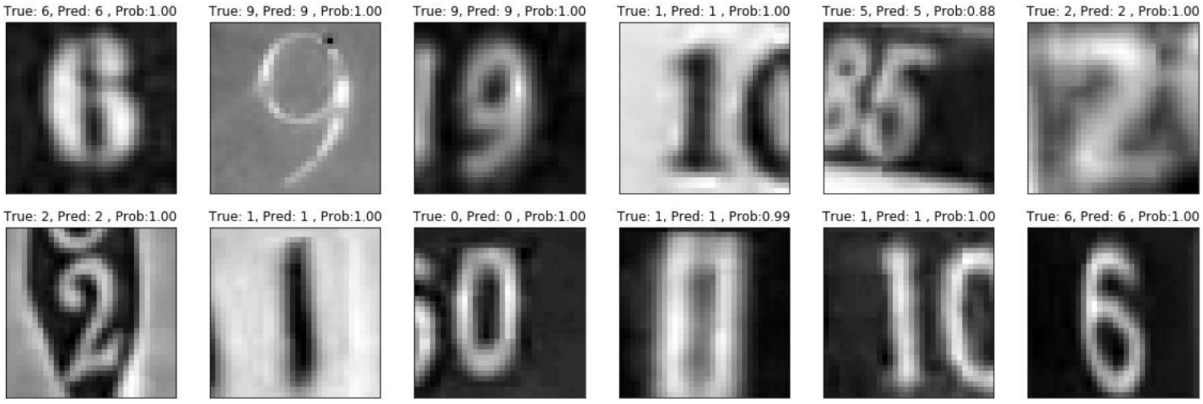
Performance on training and validation datasets:

Training accuracy	Training loss	Validation accuracy	Validation loss	Epoch
0.984672	0.056023	0.986437	0.054529	20

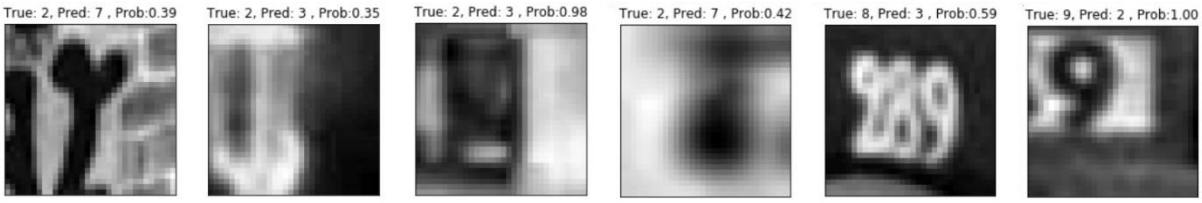
Performance on test dataset:

Loss	Accuracy
0.1548	0.9638

Correctly predicted labels:



Wrongly predicted labels:



References:

https://www.tensorflow.org/tutorials/keras

https://github.com/thePetrMarek/SequenceOfDigitsRecognition

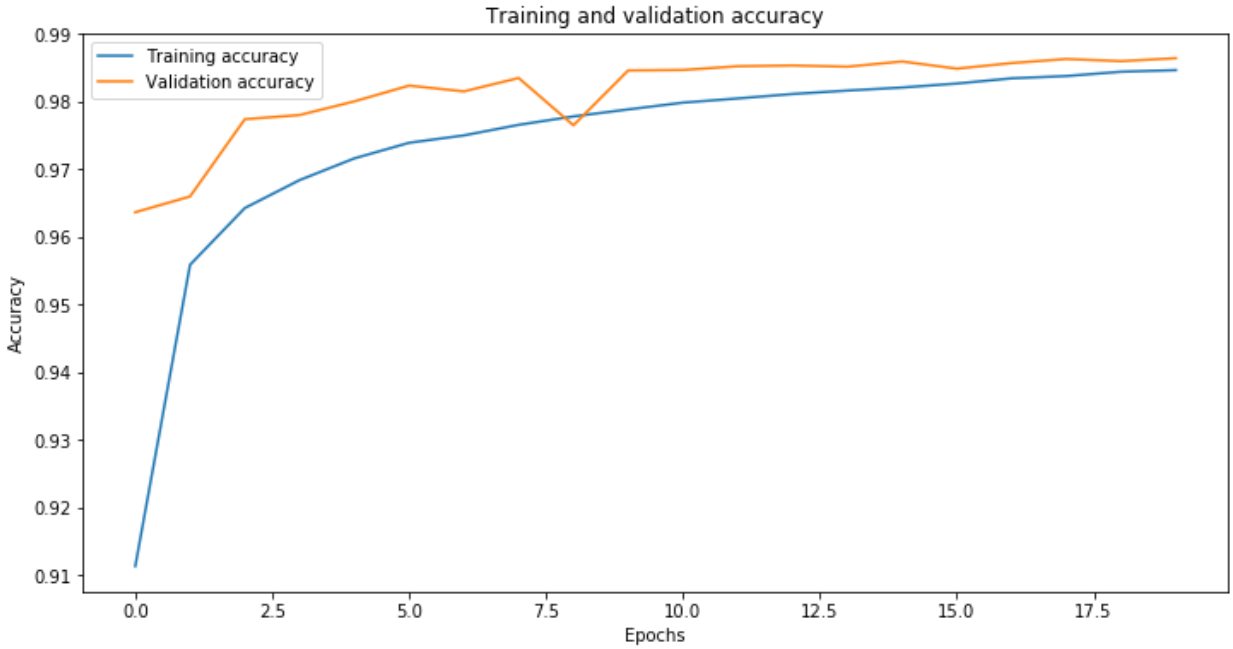
https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in-keras-329fbbadc5f5

https://ekababisong.org/gcp-ml-seminar/tensorflow/

https://github.com/auygur/CNN-SVHN_Keras

Appendix:

Training and validation datasets accuracy:



Training and validation datasets crossentropy loss