Machine Learning Challenge

Individual Contributions

Task 1	Task 2	Report
Preprocessing: Hyun Seon Initial model: Lisa, Simple CNN: Hyun Seon Complex CNN & final tuning: Mantas	Noise: Hyun Seon Top_Five_Accuracy: Lisa Sliding window: Lisa Get_predictions: Hyun Seon	Mantas

Introduction

The following report focuses on machine learning techniques to train and predict handwritten English letters from EMNIST dataset in task 1 and use the trained classifier to predict a series of letters in images in task 2. Consequently, the team has decided to use artificial neural networks as it is one the most established applications for recognition of optical characters (Araokar, 2005).

Task 1

First, a deep neural network learning model was initiated with one reshaping layer 'flatten', two 'Dense' layers with activation function 'relu' and an output layer with an activation function 'softmax' (Shevcova, 2019) which is used at multiclass classification problems. 'Tanh' activation function was initially used but resulted in lower accuracy. After fitting the model with 10 epochs and default batch size, it has produced a test accuracy of 89%. However, the training loss was significantly lower than validation loss, which suggests that the model was overfitting. The team has decided to create a more accurate model with better fit. Consequently, a convolutional neural network (CNN) algorithm was chosen.

CNN has become one of the leading techniques used for classification of contextual data. It has significant capabilities to learn contextual features and overcomes many of the issues involved in pixel-wise classification (Indolia et al., 2018). Before using a CNN, the following steps have been taken to preprocess the data.

1. Preprocessing

Partitioning the dataset. The data is split into training, validation and testing.

Transforming values. Before the transformation, the target values ranged from 1 to 26, which was causing mismatch in our predictions as Python counts from 0. To avoid this, 1 is subtracted from all the target values to readjust the range to 0~ 25.

Normalizing values. Before the normalization, the range of the data is 0 to 255. Values are normalized to increase the neural network convergence (Jamal et al., 2014). After the normalization, image values range from 0 to 1.

Value Binarization. The target vector is converted into a binary matrix representation of the input using keras's 'to_categorical' in order to be used with 'categorical_crossentropy' loss function. Since

machine learning algorithms cannot directly process categorical data, these must be converted into binary vectors which are more expressive (Jason Brownlee, 2017).

2. Building the Model

Initially, a CNN model used to predict characters was built with two 'Conv2D' layers, one reshaping layer 'flatten', two 'MaxPooling2D' layers, one 'Dropout' layer and an output layer with an activation function 'softmax'. However, although the model fit was better, the test accuracy also amounted to 89.5%.

To improve this, the team has decided to use LeNet-5 CNN architecture, with a code that optimizes for accuracy, while minimizing computational complexity (Rizwan, 2018; Deotte, 2018).

The typical LeNet-5 architecture consists of two sets of convolutional and average pooling layers, followed by a flattening convolutional layer, then two fully connected layers and finally a softmax classifier (Rizwan, 2018). The classic model was adjusted as follows:

1. Two stacked 3x3 filters substitute the single 5x5 filters, which turn into non-linear convolutions.
2. The convolution with stride 2 replaces pooling layers and these turn into learnable pooling layers.
3. 'Sigmoid' activation is replaced with 'relu'.
4. The convolution with stride 2 replaces pooling layers and these turn into learnable pooling layers.
5. Batch normalization is used to reduce internal covariate shift (Ioffe and Szegedy, 2015).
6. A dropout layer is added, which helps to prevent overfitting (Keras, 2020).
7. The softmax layer calculates a probability distribution over the 26 letters and the letter that is assigned the maximum probability is the model's output.
8. The number of nodes within each layer was based on experiments which were conducted in the framework of LeNet-5 architecture to increase efficiency.

Parameters for training of the model are passed with the 'compile" function. Optimizer 'adam' is used as it is computationally efficient, well-suited for large data and invariant to diagonal rescaling of gradients (Kingma & Ba, 2017). Loss is calculated using 'categorical_crossentropy' as there are more than two label classes and they are presented in 'one_hot' encoding. Metrics is set to 'accuracy'. The model is executed with 10 epochs and default batch size. The number of epochs were tuned based on the visual representation of train-test accuracy and loss. As the epochs get closer to 10, the model's generalizability improves. A higher number of epochs would yield higher prediction accuracy but it is set at 10 for computational efficiency.

The validation and training data sets indicate much better results and suggest that the model fit is improved. The results suggest a train, validation and test accuracy of 93.7%, 94.4% and 94.5% respectively which indicates a significant improvement on the models described before. In order to better assess model performance, *multiclass_confusion_matrix* and *class_scores* functions were made that examine other evaluation metrics such as recall and precision, and the performance per class.

Task 2

Task 2 involved new images, each including a sequence of five handwritten letters, to be classified using the model of Task 1 which makes this into a multi-label classification problem. These new images included noise and the handwritten letters were more illegible and unevenly spaced. Therefore, to make a more robust model, salt and pepper noise with amount 0.4 was added to the original image to which the model was fitted. The validation and test accuracy resulted in 94.4% and 94.7% respectively with 10 epochs. Additionally, test images were resized from 30 x 168 to 28 x 168 by deleting the first and last row.

In order to navigate the sequence of letters, the sliding window method was implemented to divide each test image into segments of size 28 x 28 to match the dimensions of the training images. The function *strided_axis0* (Stack Overflow, 2018) used strides to cut the larger image into 140 segments starting from the left and moving along axis one stepping one column further at each iteration.

From these 140 segments, different combinations had to be chosen to best capture the five letters on the test image. This would allow for calculating the label for each segment. To that end, the function *top_five_accu* was created with the arguments *image*, indicating test image to be classified, *start*, indicating the starting column of the first segment, and *step*, indicating how many columns after *start* the next segment should be placed. After several attempts, the *step* range with the best results varied between 5 to 20 and the *start* range between 0 to 10.

Top_five_accu classifies the chosen segments using Sequential's predict method which generates a probability distribution of 26 scores for each segment. The highest of the 26 scores is chosen and a sparse matrix is created to represent this particular letter, which is then converted into the corresponding number. Next, a dictionary is created with the highest scores of each segment as keys and the numbers as values. From the dictionary, the values of the five highest scores were chosen using *nlargest* function from the Collections module. The function outputs these five values as a list.

The above process was then repeated several times using different *step* sizes and *start* columns for the same test image. The function *get_predictions* was created for that purpose. Its only argument is the test-image to be classified. *Get_predictions* runs the previous function (*top_five_accu*) several times with different arguments, transforms its outputs (i.e. predictions) into strings, and adds them into a dictionary. A certain prediction might occur multiple times, therefore, the keys in the dictionary correspond to the predictions and the values are the number of times each prediction occurred. After the dictionary is sorted by frequency, the five predictions with the largest frequencies are returned. Finally, the hamming distance is used to compare these five predictions with the actual one and decide on the winner-label. Based on the four true labels provided in the assignment, the results were the following (*Y true* = true label, *Y predicted* = label predicted by the model, *Image Index*: number of test image):

Table 1: Classification of the four test-images mentioned in the assignment description

Image Index	Top 5 predictions	Y true	Y predicted
0	['0404262525', '0404022625', '0412042625', '1204022512', '2312042625']	0412042625	0412042625
1	['2203262612', '2222262626', '2226261012', '1403261212', '2211092612']	2203261217	2203262612
-2	['0606070303', '0621140703', '0621070303', '0621070312', '0621170703']	0612010703	0606070303
-1	['2020021002', '2002021002', '2002171002', '2002100210', '2025021002']	2002151002	2002171002

References

Araokar, S. (2005). Visual Character Recognition using Artificial Neural Networks. *ArXiv:Cs/0505016*. http://arxiv.org/abs/cs/0505016

Brownlee, J. (2017, July 11). How to One Hot Encode Sequence Data in Python. *Machine Learning Mastery*. https://machinelearningmastery.com/how-to-one-hot-encode-sequence-data-in-python/

Chollet, F., & others. (2015). Keras. GitHub. Retrieved from https://github.com/fchollet/keras

Cohen, G., Afshar, S., Tapson, J., & van Schaik, A. (2017). EMNIST: An extension of MNIST to handwritten letters. *ArXiv:1702.05373 [Cs]*. http://arxiv.org/abs/1702.05373

Deotte, C. (2018). *How to choose CNN Architecture MNIST*. https://kaggle.com/cdeotte/how-to-choose-cnn-architecture-mnist

Indolia, S., Goswami, A. K., Mishra, S. P., & Asopa, P. (2018). Conceptual Understanding of Convolutional Neural Network- A Deep Learning Approach. *Procedia Computer Science*, *132*, 679–688. https://doi.org/10.1016/j.procs.2018.05.069

Hunter, D.. (2007). Matplotlib: A 2D Graphics Environment, Computing in Science & Engineering, 9, 90-95

Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *ArXiv:1502.03167 [Cs]*. http://arxiv.org/abs/1502.03167

Keras, K. (2020). *Keras documentation: The Sequential model*. https://keras.io/guides/sequential_model/

Kingma, D. P., & Ba, J. (2017). Adam: A Method for Stochastic Optimization. *ArXiv:1412.6980 [Cs]*. http://arxiv.org/abs/1412.6980

McKinney, W., & others. (2010). Data structures for statistical computing in python. In Proceedings of the 9th Python in Science Conference (Vol. 445, pp. 51–56).

Oliphant, T. E. (2006). A guide to NumPy (Vol. 1). Trelgol Publishing USA.

Oliphant, E.. (2007). Python for Scientific Computing, Computing in Science & Engineering, 9, 10-20, DOI:10.1109/MCSE.2007.58

Peshawa Jamal, Ali, M., Rezhna Hassan Faraj, Peshawa J Muhammad Ali, & Rezhna H Faraj. (2014). *Data Normalization and Standardization: A Technical Report*.
https://doi.org/10.13140/RG.2.2.28948.04489

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.. (2011). Scikit-learn: Machine Learning in Python, Journal of Machine Learning Research, 12, 2825-2830

Rizwan, M. (2018, September 30). LeNet-5—A Classic CNN Architecture. *EngMRK*. https://engmrk.com/lenet-5-a-classic-cnn-architecture/

Shevcova, K. (2019). *Handwritten letter recognition with a flick*. https://kaggle.com/swordey/handwritten-letter-recognition-with-a-flick

Xu, Y., & Goodacre, R. (2018). On Splitting Training and Validation Set: A Comparative Study of Cross-Validation, Bootstrap and Systematic Sampling for Estimating the Generalization Performance of Supervised Learning. *Journal of Analysis and Testing*, *2*(3), 249–262. https://doi.org/10.1007/s41664-018-0068-2

Van der Walt, S., Schönberger, J., Nunez-Iglesias, J., Boulogne, F., D. Warner, J., Yager, N., Gouillart, E., Yu, T.. (2014). scikit-image: image processing in Python, PeerJ 2:e453

Unknown, (2018). Sliding windows from 2D array that slides along axis=0 or rows to give a 3D array. Retrieved from: https://stackoverflow.com/questions/43185589/sliding-windows-from-2d-array-that-slides-along-axis-0-or-rows-to-give-a-3d-arra