Yanisa Sunthornyotin

# Computational Neuroscience:Final Project (option B)

Neural network algorithm: Convolution Neural Network
Data: MNIST Cyrillic (Handwriting data) ex.

Tools: Tensorflow/Jupyter Notebook
Technical approach:

- Sampling input data / encode the categorical label to numerical label
- Split data into Train/Test set
- Tune the parameter layer to match the desired input
- Evaluate the model to observe accuracy

Ref:https://www.tensorflow.org/tutorials/images/cnn
https://github.com/GregVial/CoMNIST/blob/master/images/Cyrillic.zip

From .png picture file in folder Cyrillic, firstly collect and transform each file into greyscale and to lower dimension (low resolution) which is 28x28 instead of 278x278 to reduce the processing workload. Since the picture itself doesn't require high resolution in order to categorize the handwriting, reducing the dimension of the picture to 28x28 is still acceptable.
Here is the number of all training picture in each label category. Each picture in the same category has a slightly different shape and angle but still, be able to be recognized as one of that label.

```
{'Б': 444, 'Ф': 463, 'Ж': 462, 'У': 551, 'Ъ': 459, 'П': 474, 'Э': 458, 'И': 447, 'Ь': 427, 'Ы': 415, 'О': 465,
'З': 431, 'Т': 456, 'А': 469, 'Х': 486, 'Ё': 344, 'Й': 466, 'І': 247, 'Е': 576, 'Р': 493, 'В': 487, 'Ч': 464, 'Ю':
461, 'Л': 433, 'Щ': 446, 'М': 470, 'Ш': 431, 'Н': 508, 'Я': 438, 'К': 459, 'Г': 424, 'Ц': 448, 'Д': 465, 'С': 513}
```

Now, select only the first 200 handwriting sample from each category to make the overall combining training/testing dataset more uniform.

```
{'Б': 200, 'Ф': 200, 'Ж': 200, 'У': 200, 'Ъ': 200, 'П': 200, 'Э': 200, 'И': 200, 'Ь': 200, 'Ы': 200, 'О': 200,
'З': 200, 'Т': 200, 'А': 200, 'Х': 200, 'Ё': 200, 'Й': 200, 'І': 200, 'Е': 200, 'Р': 200, 'В': 200, 'Ч': 200, 'Ю':
200, 'Л': 200, 'Щ': 200, 'М': 200, 'Ш': 200, 'Н': 200, 'Я': 200, 'К': 200, 'Г': 200, 'Ц': 200, 'Д': 200, 'С': 200}
```

Type of dataset

We encode the label of character into numerical according to the index in the key array in order to make it processable by the model.
For example, Б=0, Ф=1, and so on

By typing this command

```
train_images, test_images, train_labels, test_labels = train_test_split(np.array(X), np.array(y),
                                                    test_size=0.2,
                                                    random_state=0)
```

We split training and testing data where the training set is 80% of all data and the testing set is 20% of all data. Each image is collected in the form of a matrix where each element ranging from 0 to 1 since it is greyscaled.

```
classes: ['Б', 'Ф', 'Ж', 'У', 'Ъ', 'П', 'Э', 'И', 'Ь', 'Ы', 'О', 'З', 'Т', 'А', 'Х', 'Ё', 'Й', 'І', 'Е', 'Р',
'В', 'Ч', 'Ю', 'Л', 'Щ', 'М', 'Ш', 'Н', 'Я', 'К', 'Г', 'Ц', 'Д', 'С']
number of category: 34
train image collection: <class 'numpy.ndarray'>
train image: <class 'numpy.ndarray'>
training label collection: <class 'numpy.ndarray'>
training label: <class 'numpy.ndarray'>
shape of training image: (28, 28, 2)
```

The training dataset and testing dataset are collected in the same format.

As input, a CNN takes tensors of shape (image_height, image_width, color_channels), ignoring the batch size then passing the argument input_shape to the first layer. For this dataset, those tensor is (28,28,2) different from the example where input_shape is (32,32,3) and process along with RGB format.

Model Evaluation / Result

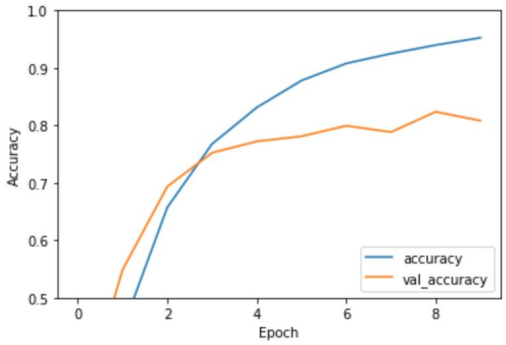
To create a convolutional neural network model, we define each layer which some will transform and convolute the data matrix and some will extract out the characteristic of the matrix. This is a process of learning where we input x, train w and output as y.
The width and height dimensions tend to decrease as we go deeper into the network. According to the documentation, as the width and height shrink, we are still able to add more output channels in each Conv2D layer.

In this project, we tried to experiment on a different type of layer, adding various kind of activation function ex. ReLu and Softmax to see the change in the evaluation result.

To complete our model, we feed the last output tensor from the convolutional base into one or more Dense layers to perform classification. However, in order to use dense layer, we must flatten the matrix first then dense layer will output 34 classes of the classification result.

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 26, 26, 32)        608
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 3, 3, 64)          36928
_________________________________________________________________
flatten_6 (Flatten)          (None, 576)               0
_________________________________________________________________
dense_12 (Dense)             (None, 64)                36928
_________________________________________________________________
dense_13 (Dense)             (None, 34)                2210
=================================================================
Total params: 95,170
Trainable params: 95,170
Non-trainable params: 0
_________________________________________________________________
```

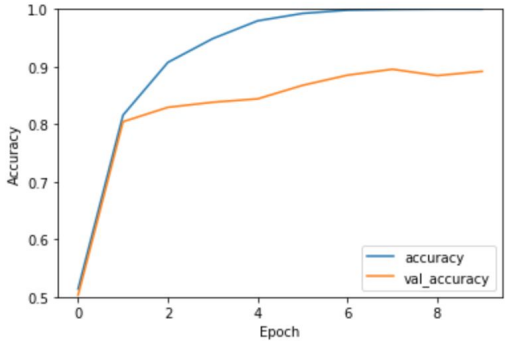
```
test accuracy: 0.80808824
1360/1 - 0s - loss: 0.7672 - accuracy: 0.8081
```




From the first neural network architecture, we plot the evaluation graph where we applied the model to the testing dataset. As a result, the model gives up to 80% of accuracy in categorizing the label. While the following architecture which contain a larger number of hidden layers gives the higher accuracy score up to 90% when applying to the same testing data. We also add batch normalization to each layer which actually can improve the model accuracy.

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_16 (Conv2D)           (None, 26, 26, 32)        608
_________________________________________________________________
batch_normalization (BatchNo (None, 26, 26, 32)        128
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 11, 11, 64)        18496
_________________________________________________________________
batch_normalization_1 (Batch (None, 11, 11, 64)        256
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 3, 3, 64)          36928
_________________________________________________________________
batch_normalization_2 (Batch (None, 3, 3, 64)          256
_________________________________________________________________
flatten_7 (Flatten)          (None, 576)               0
_________________________________________________________________
dense_14 (Dense)             (None, 64)                36928
_________________________________________________________________
dense_15 (Dense)             (None, 34)                2210
=================================================================
Total params: 95,810
Trainable params: 95,490
Non-trainable params: 320
_________________________________________________________________
```

```
1360/1 - 1s - loss: 0.6705 - accuracy: 0.8919
test accuracy: 0.89191175
```




In conclusion, the model can be improved as we add more layer or change the type of activation function. However, the efficiency of changing each activation function and adding the depth of the neural network also depends on the input data and the category we want to classify. This Convolutional neural network can also be applied to other kinds of dataset ex. Object image identification, NLP, recommender systems, etc.