Optical character recognition

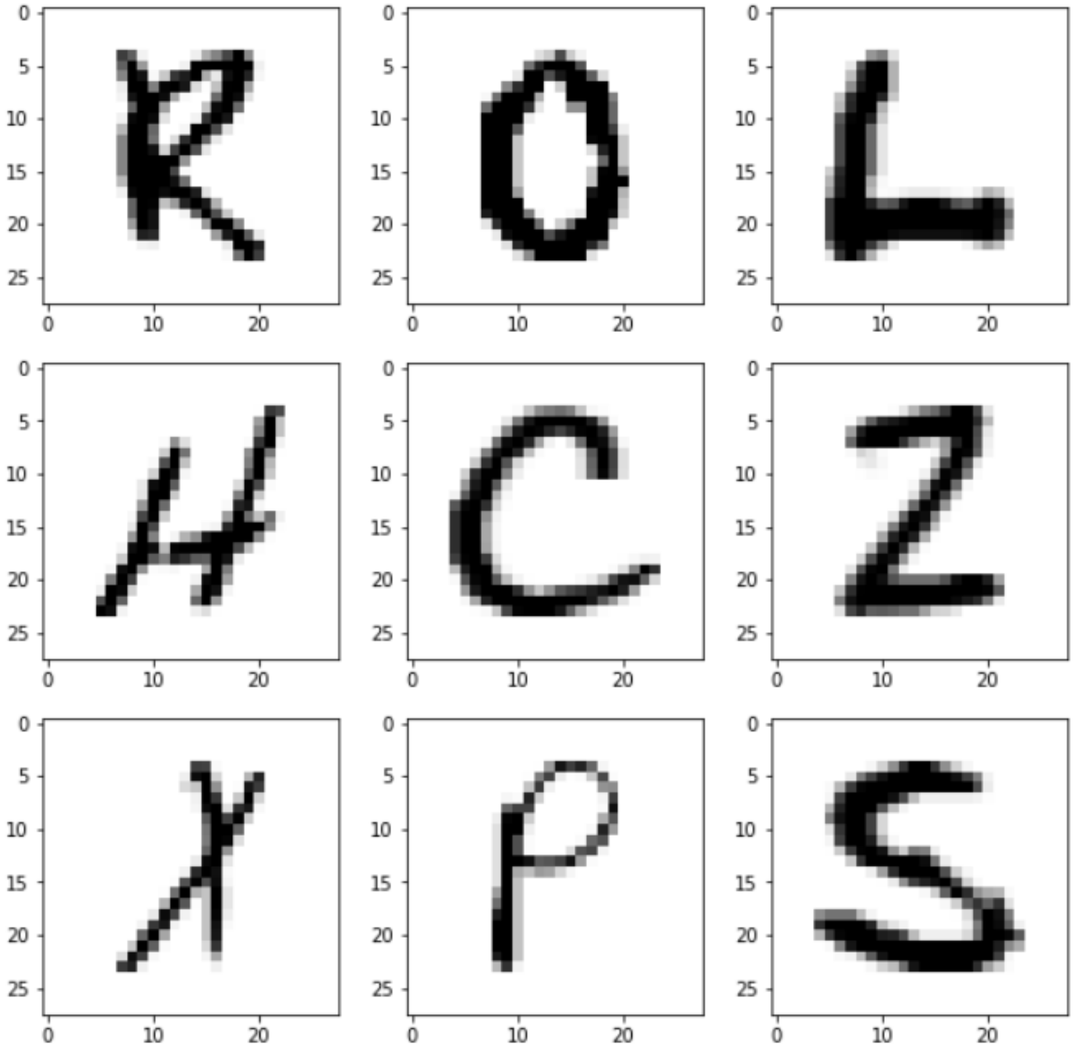
Виконав

Левадний Микола, МІ-4

Dataset

A-Z Handwritten Alphabets in .csv format

The dataset contains 26 folders (A-Z) containing handwritten images in size 28x28 pixels, each alphabet in the image is centre fitted to 20x20 pixel box.

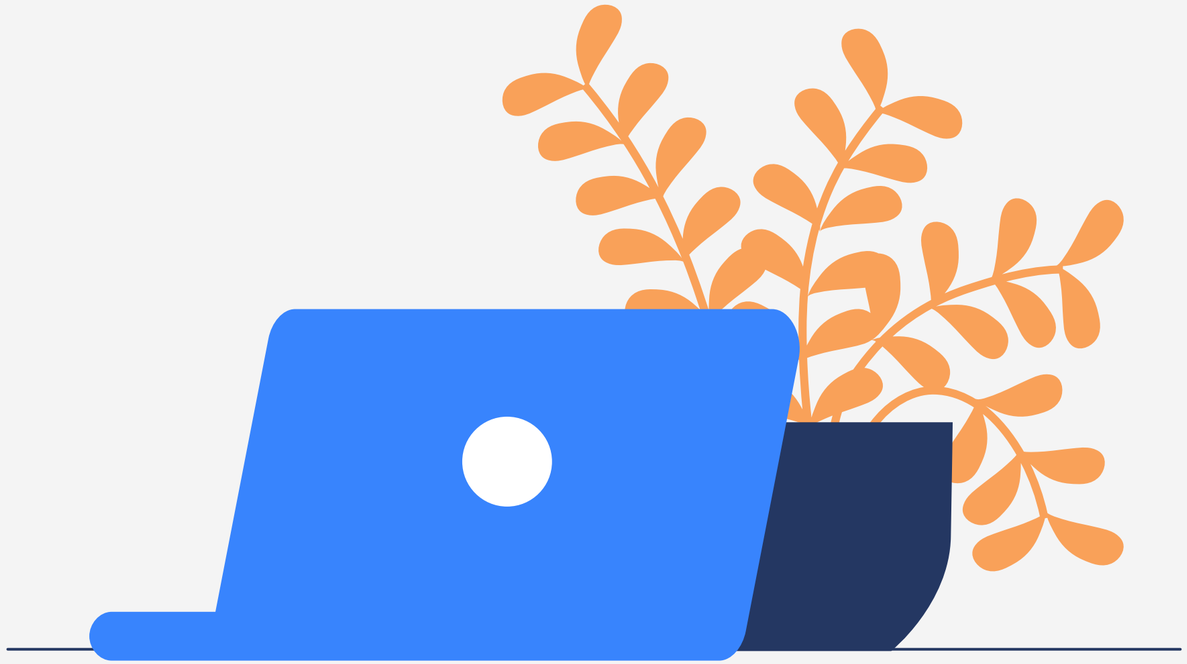


Add data augmentation

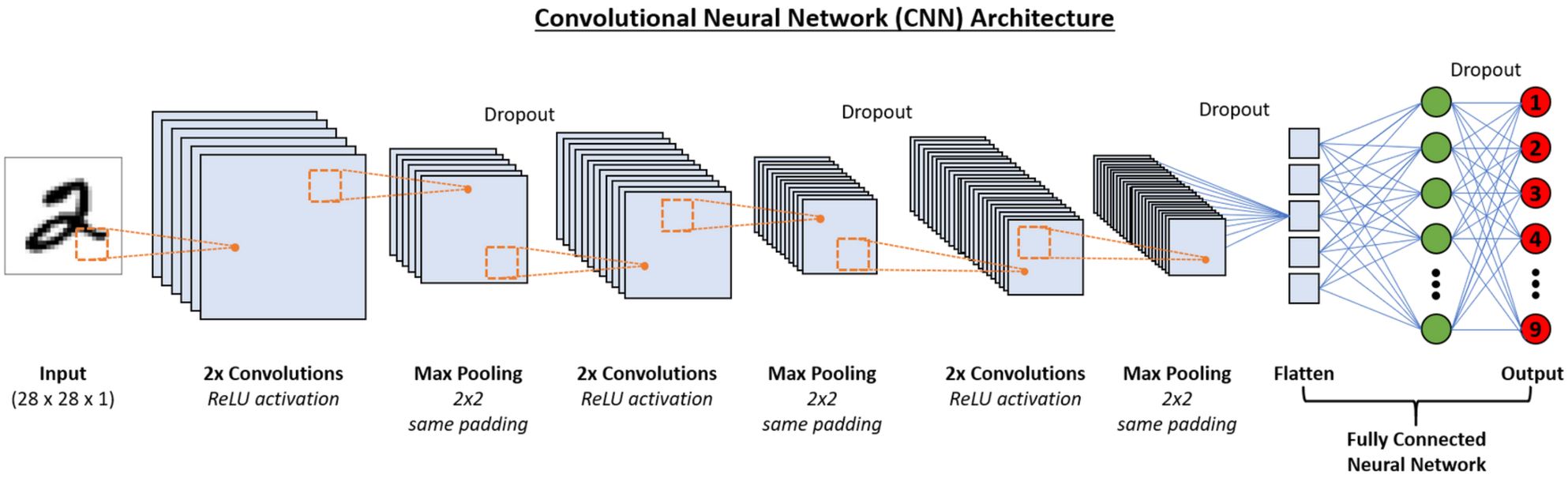
```
data_augmentation = tf.keras.Sequential([
  layers.experimental.preprocessing.RandomRotation(0.08),
  layers.experimental.preprocessing.RandomContrast(0.12),
  layers.experimental.preprocessing.RandomZoom(height_factor=(0.05,0.1))
])
```

Augmentation allows you to expand the dataset with the help of already received data

- Small rotation (prevent letter overturn)
- Changing contrast of image
- Zooming



CNN model



A Convolutional Neural Network, also known as CNN or ConvNet, is a class of neural networks that specializes in processing data that has a grid-like topology, such as an image. A digital image is a binary representation of visual data. It contains a series of pixels arranged in a grid-like fashion that contains pixel values to denote how bright and what color each pixel should be.

CNN realization code snippet

(using keras)

```
[39] model = Sequential()
     model.add(data_augmentation)
     #CNN
     # input -> conv -> maxpool -> conv -> maxpool .....->flattened vector->
     #.                        hidden layer -> hidden layer -> softmax layer
     model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(28,28,1)))
     model.add(MaxPool2D(pool_size=(2, 2), strides=2))

     model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding = 'same'))
     model.add(MaxPool2D(pool_size=(2, 2), strides=2))

     model.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding = 'valid'))
     model.add(MaxPool2D(pool_size=(2, 2), strides=2))

     model.add(Flatten())

     model.add(Dense(64,activation ="relu"))
     model.add(Dense(128,activation ="relu"))

     model.add(Dense(26,activation ="softmax"))
```

```
model.summary()
```

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential_9 (Sequential)   (None, 28, 28, 1)         0

 conv2d_20 (Conv2D)          (None, 26, 26, 32)        320

 max_pooling2d_19 (MaxPoolin  (None, 13, 13, 32)       0
 g2D)

 conv2d_21 (Conv2D)          (None, 13, 13, 64)        18496

 max_pooling2d_20 (MaxPoolin  (None, 6, 6, 64)         0
 g2D)

 conv2d_22 (Conv2D)          (None, 4, 4, 128)         73856

 max_pooling2d_21 (MaxPoolin  (None, 2, 2, 128)        0
 g2D)

 flatten_7 (Flatten)         (None, 512)               0

 dense_20 (Dense)            (None, 64)                32832

 dense_21 (Dense)            (None, 128)               8320

 dense_22 (Dense)            (None, 26)                3354

=================================================================
Total params: 137,178
Trainable params: 137,178
Non-trainable params: 0
_________________________________________________________________
```

Training results

Compiling the Model

```
[61] model.compile(optimizer = Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
```

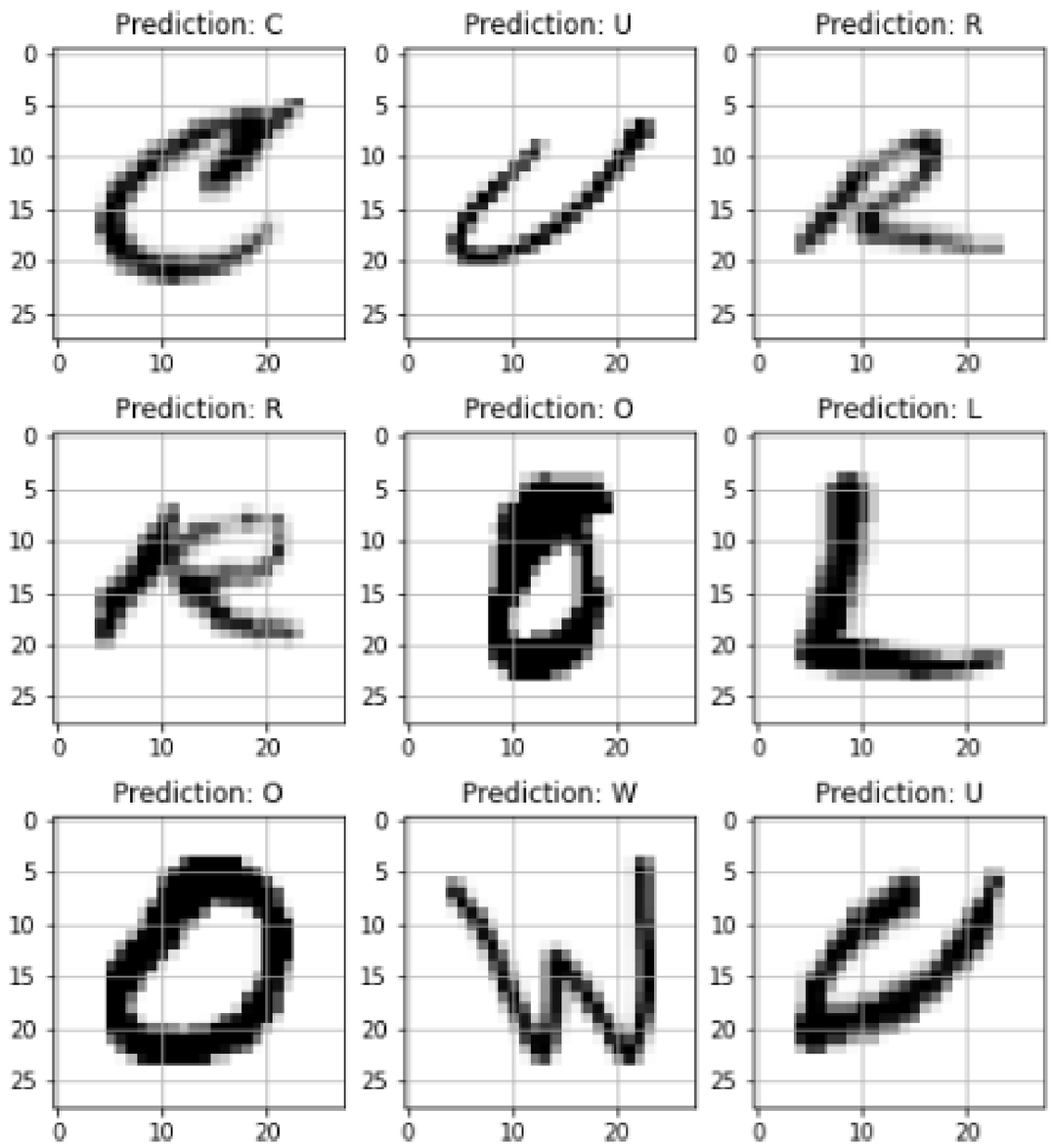
0 сек.

Starting the Training

```
[62] history = model.fit(X_train, train_yOHE, epochs=10,  validation_data = (X_test,test_yOHE))
```

6 мин.

```
Epoch 1/10
9312/9312 [==============================] - 40s 4ms/step - loss: 0.1580 - accuracy: 0.9564 - val_loss: 0.0951 - val_accuracy: 0.9725
Epoch 2/10
9312/9312 [==============================] - 38s 4ms/step - loss: 0.0713 - accuracy: 0.9801 - val_loss: 0.0677 - val_accuracy: 0.9822
Epoch 3/10
9312/9312 [==============================] - 37s 4ms/step - loss: 0.0597 - accuracy: 0.9835 - val_loss: 0.0753 - val_accuracy: 0.9810
Epoch 4/10
9312/9312 [==============================] - 37s 4ms/step - loss: 0.0557 - accuracy: 0.9850 - val_loss: 0.0650 - val_accuracy: 0.9839
Epoch 5/10
9312/9312 [==============================] - 43s 5ms/step - loss: 0.0513 - accuracy: 0.9859 - val_loss: 0.0662 - val_accuracy: 0.9824
Epoch 6/10
9312/9312 [==============================] - 38s 4ms/step - loss: 0.0503 - accuracy: 0.9867 - val_loss: 0.0591 - val_accuracy: 0.9853
Epoch 7/10
9312/9312 [==============================] - 38s 4ms/step - loss: 0.0491 - accuracy: 0.9872 - val_loss: 0.0655 - val_accuracy: 0.9846
Epoch 8/10
9312/9312 [==============================] - 37s 4ms/step - loss: 0.0498 - accuracy: 0.9874 - val_loss: 0.0738 - val_accuracy: 0.9833
Epoch 9/10
9312/9312 [==============================] - 37s 4ms/step - loss: 0.0468 - accuracy: 0.9884 - val_loss: 0.0680 - val_accuracy: 0.9856
Epoch 10/10
9312/9312 [==============================] - 37s 4ms/step - loss: 0.0501 - accuracy: 0.9880 - val_loss: 0.0642 - val_accuracy: 0.9857
```



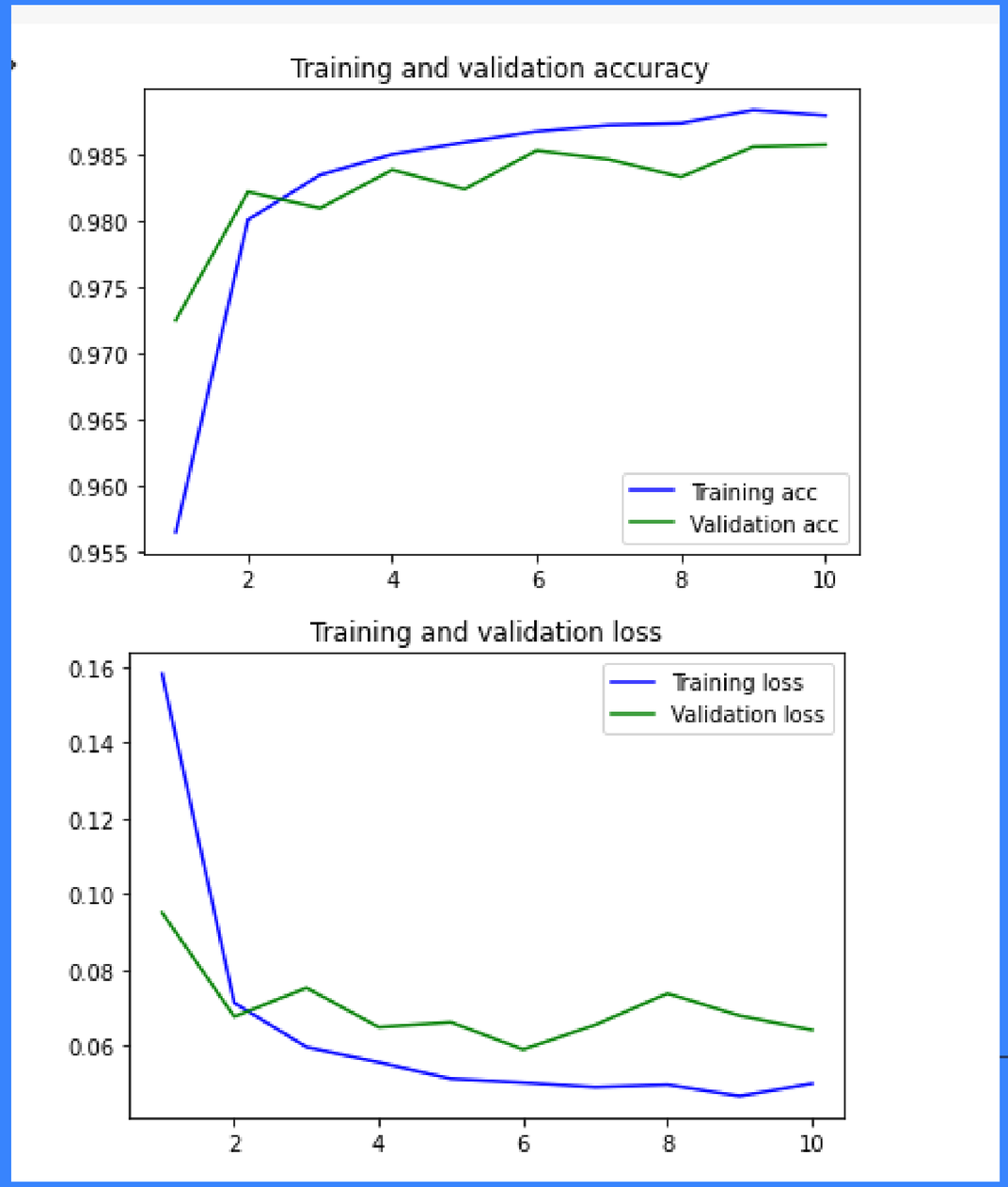
Test on validation set

Accuracy result: 0.9857

Making Predictions

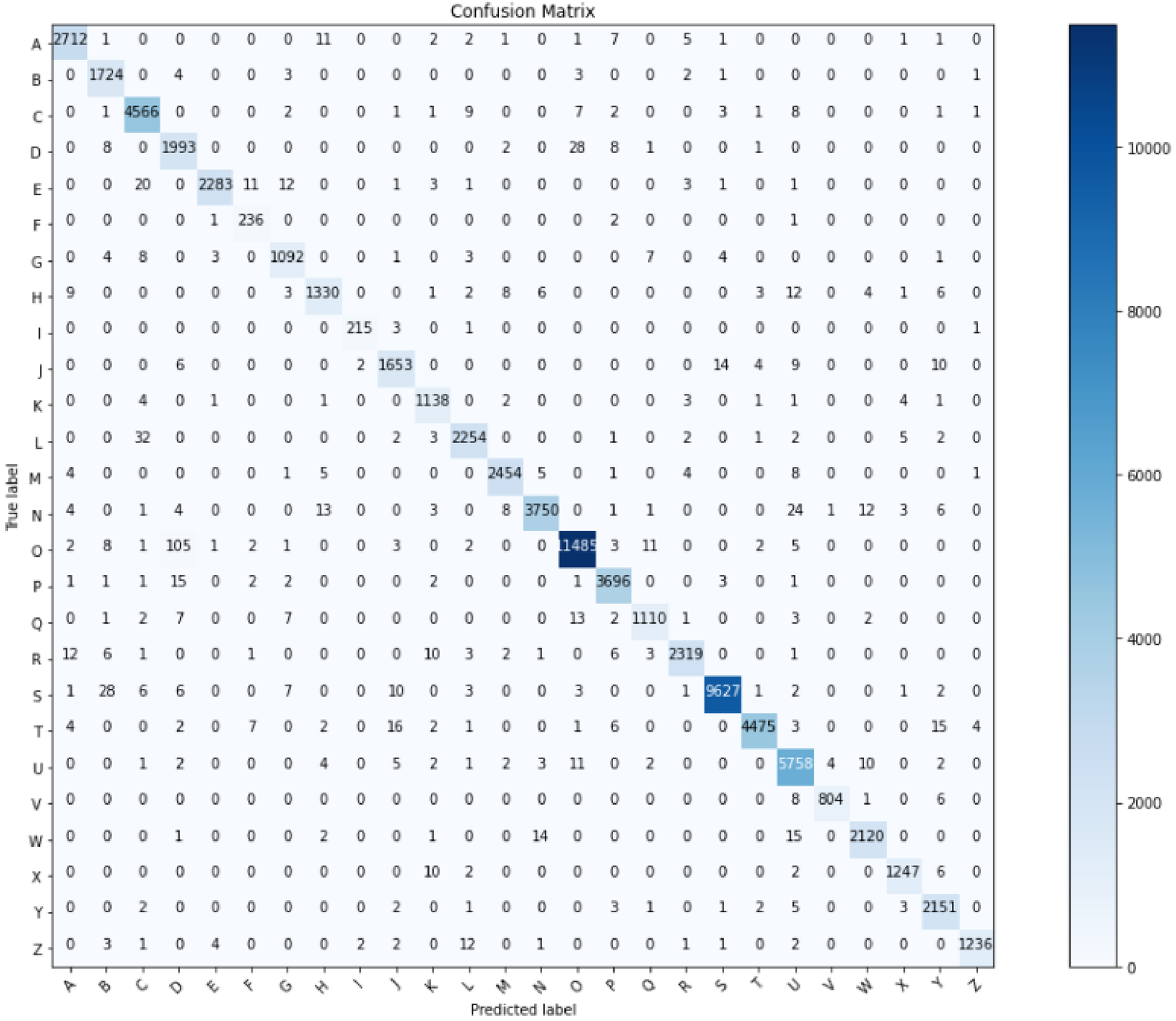
```
[63] model.evaluate(X_test, np.array(test_yOHE))

     2328/2328 [==============================] - 6s 3ms/step - loss: 0.0642 - accuracy: 0.9857
     [0.06420440971851349, 0.9857430458068848]
```



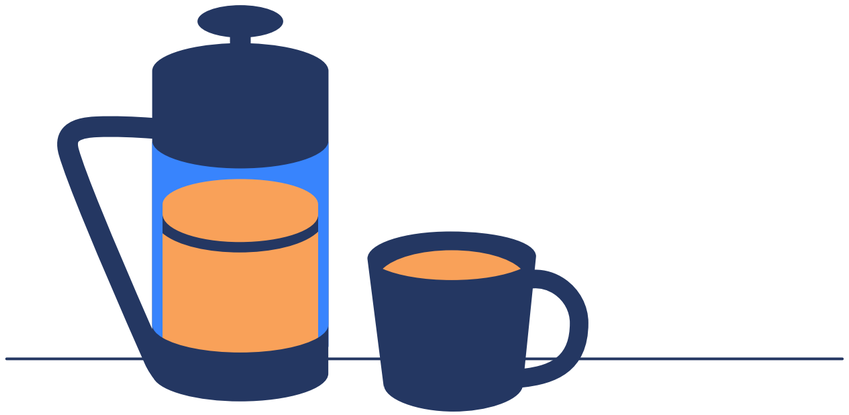
Training and validation accuracy and loss trend

Confusion Matrix



Typicall errors

We can notice that the model is trained well enough, that's why there are not many errors. But still there are enough cases where there were mix-ups O-D and N-H



Thanks for attention!