

A thick dark blue vertical bar runs down the left side of the slide. A blue arrow-shaped banner points to the right from this bar, containing the date. Below the bar, several thin, curved lines in dark blue and light grey sweep upwards and to the right.

8/3/2022

Deep learning – KMNIST

Olivier FONTAINE

TABLE OF CONTENTS

Analysis objective.....	2
Data set description	2
Data exploration, data cleaning & feature engineering.....	3
Data Exploration	4
Feature engineering.....	4
Preprocessing	4
Training & validation.....	5
Training DEEP LEARNING models.....	6
Fully Connected Neural Network.....	7
Convolutional Neural Network (CNN)	9
CNN ResNet-34 Neural Network	12
DEEP LEARNING Model recommendation.....	14
Key Findings and Insights	14
Suggestions for next steps in analyzing this data	15
Appendix: code.....	15

ANALYSIS OBJECTIVE

- ➔ *Main objective of the analysis that also specifies whether your model will be focused on a specific type of Deep Learning or Reinforcement Learning algorithm and the benefits that your analysis brings to the business or stakeholders of this data.*

The objective of this analysis is to use a set of deep learning methods to classify correctly handwritten Japanese hiragana character.

So, this is a classification problem on computer vision data.

If results are good enough, the data stakeholders might extend the project to all hiragana characters (there are 46 basic hiragana characters) as well as all katakana characters (2nd Japanese alphabet, 46 basic katakana characters).

This can also be extended to the 3rd type of written Japanese characters known as kanjis, this might be a more complex project as there are at least 3,000 kanjis in Japanese. But, if all of this is done, this could be used to translate old handwritten Japanese documents in another language. A recurrent neural network (RNN) might be added to understand whole sentence meaning (context).

DATA SET DESCRIPTION

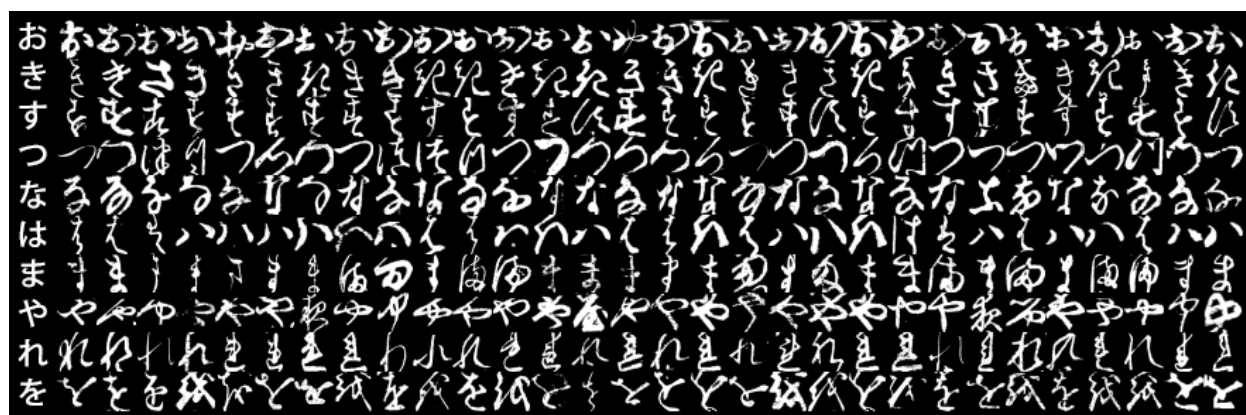
- ➔ *Brief description of the data set you chose, a summary of its attributes, and an outline of what you are trying to accomplish with this analysis.*

The data set name is **Kuzushiji-MNIST** Dataset.

The data set is available on KAGGLE: <https://github.com/rois-codh/kmnist>

The data set has **70,000** grayscales images (28*28) spanning 10 classes (one from each column of hiragana character).

Here are the 10 classes of Kuzushiji-MNIST, with the first column showing each character's modern hiragana counterpart.



index	codepoint	char
0	0	U+304A お
1	1	U+304D き
2	2	U+3059 す
3	3	U+3064 つ
4	4	U+306A な
5	5	U+306F は
6	6	U+307E ま
7	7	U+3084 や
8	8	U+308C れ
9	9	U+3092 を

The dataset is divided in 4 files:

File	Number of images
Training images	60,000
Training labels	60,000
Testing images	10,000
Testing labels	10,000

The dataset is balanced for both the training and the test set:

File	Number of images	Number of images per class
Training images	60,000	6,000
Training labels	60,000	6,000
Testing images	10,000	1,000
Testing labels	10,000	1,000

So, in this project, we will try different deep learning techniques in order to classify correctly the images within the 10 identified hiragana characters. We will evaluate those techniques with the accuracy metric as the dataset is balanced.

DATA EXPLORATION, DATA CLEANING & FEATURE ENGINEERING


➔ *Brief summary of data exploration and actions taken for data cleaning and feature engineering.*

DATA EXPLORATION

Below, you can find samples from the training set with their label on the right.

I added on the left the document that was provided with the dataset that map the label with the hiragana character:

index	codepoint	char	
0	0	U+304A	お
1	1	U+304D	き
2	2	U+3059	す
3	3	U+3064	つ
4	4	U+306A	な
5	5	U+306F	は
6	6	U+307E	ま
7	7	U+3084	や
8	8	U+308C	れ
9	9	U+3092	を



FEATURE ENGINEERING

With Deep Learning methodologies for computer vision, the feature engineering is actually performed :

- within the full connected neural network for full connected neural networks
- within the convolutional part of a CNN architecture for CNN

PREPROCESSING

We need to one hot encode the labels datasets. Actually, we used the KERAS method `keras.utils.to_categorical`.

- Initially, the shape of the train labels dataset is (60000,). There only 1 column vector:

```
array([8, 7, 0, ..., 0, 4, 9], dtype=uint8)
```

- After one hot encoding:

```
: y_train
array([[0., 1., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 1., 0.],
       [1., 0., 0., ..., 0., 0., 0.]], dtype=float32)
```

We also need to reshape the numpy arrays for images to a 28*28*1 array for each image.

And finally, we need to scale our features. Each pixel of the image is a feature. As the image is in gray scale, each pixel holds a grey value from 0 to 255. In order to normalize it, we need to divide the pixel value by 255, see explanations below:

$$\text{Normalized value} = \frac{\text{Input value} - \text{Min}}{\text{Max} - \text{Min}}$$

$$\text{Normalized value} = \frac{\text{Input value} - 0}{255 - 0}$$

$$\text{Normalized value} = \frac{\text{Input value}}{255}$$

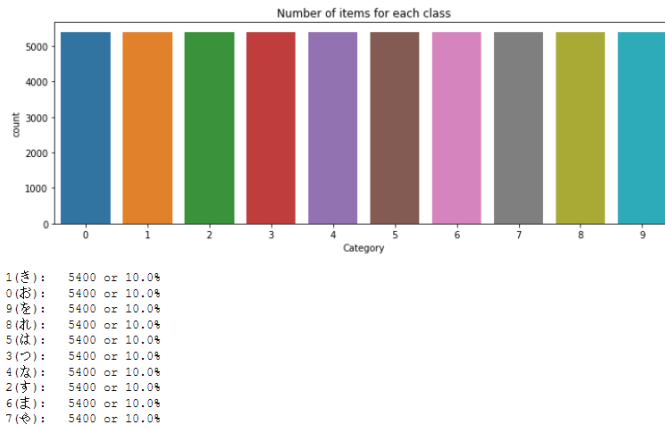
TRAINING & VALIDATION

The training set is shuffle split into a training set and a validation set:

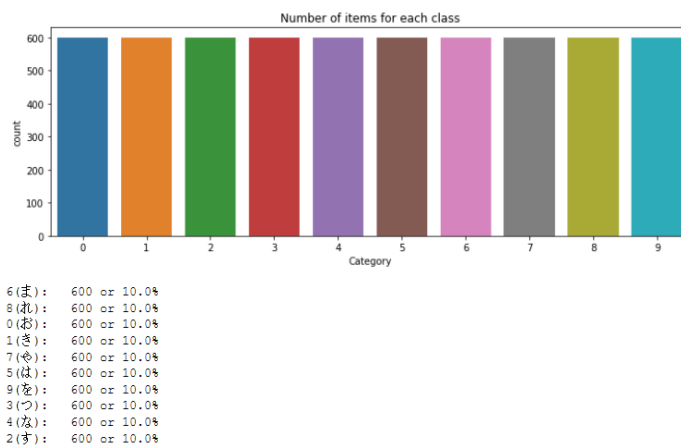
File	Number of images	Number of images per class
Training images	54,000	5,400
Training labels	54,000	5,400
Testing images	6,000	600
Testing labels	6,000	600

The validation set is used for hyper parameter tuning.

Here is the new balanced training set:



Here is the balanced validation set:



TRAINING DEEP LEARNING MODELS

➔ *Summary of training at least three variations of the Deep Learning model you selected. For example, you can use different clustering techniques or different hyperparameters.*

For this purpose, 3 neural network architectures are evaluated:

- Fully connected neural network
- A Convolutional Neural Network (CNN)
- A CNN ResNet-34 neural network

Here are the different hyperparameters that we can tune during evaluation:

- batch size
- learning rate
- optimizer

- number of epochs
- number of hidden layers
- number of neurons per layer
- activation functions

as well as those specific to CNN:

- number of filters
- filter dimension
- stride
- padding
- dropout / early stopping
- ...

FULLY CONNECTED NEURAL NETWORK

The 1st model is a fully connected neural network.

This model was manually tuned based on the evaluation of the results against the validation set (10% of the original training set).

File	Selected value	Tested values
Batch size	32	16, 32, 64
Learning rate	0.01	0.01 & 0.001
Optimizer	Adagrad	SGD, Adagrad, RMSprop, Adam
Number of epochs	20	5, 10, 20
Activation function	Relu And finally softmax	Relu & softmax
Hidden layers	2	1, 2
Number of neurons per layer	100	10, 50, 100

Surprisingly, Adagrad was a better optimizer than Adam.

Then, the number of neurons greatly affects the results contrary to the number of additional hidden layers.

Finally, the increase of the number of epochs allow a better training. I could have gone over 20 epochs.

Here is the final architecture of this neural net:

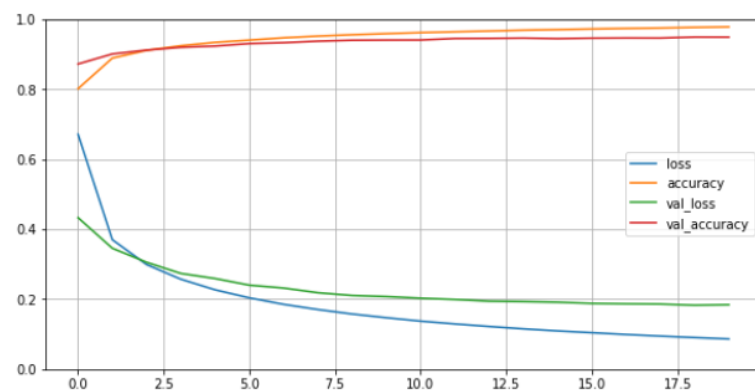

```
model1 = keras.models.Sequential()
model1.add(keras.layers.Flatten(input_shape=[IMG_ROWS, IMG_COLS]))
model1.add(keras.layers.Dense(100, activation="relu"))
model1.add(keras.layers.Dense(100, activation="relu"))
model1.add(keras.layers.Dense(10, activation="softmax"))
```

Layer (type)	Output Shape	Param #
flatten_4 (Flatten)	(None, 784)	0
dense_4 (Dense)	(None, 100)	78500
dense_5 (Dense)	(None, 100)	10100
dense_6 (Dense)	(None, 10)	1010
Total params: 89,610		
Trainable params: 89,610		
Non-trainable params: 0		

And then the results after 20 epochs.

```
Epoch 20/20
1688/1688 [=====] - 2s 1ms/step - loss: 0.0861 - accuracy: 0.9787 - val_loss: 0.1841 - val_accuracy: 0.9493
```

Let's plot the loss & accuracy for the training and the validation set:



Both accuracies and losses reached a plateau after 20 epochs.

Now, let's perform a test on the test set:

```
] : model1.evaluate(X_test, y_test)
313/313 [=====] - 0s 701us/step - loss: 0.4346 - accuracy: 0.8800
[0.434583455324173, 0.8799999952316284]
```

Here are the summarized results for this model:

Type of accuracy	Score
Training Accuracy	97.87
Validation Accuracy	94.93
Test Accuracy	87.99

➔ The test accuracy is not that good.

CONVOLUTIONAL NEURAL NETWORK (CNN)

The 2nd model is a Convolutional Neural Network.

This model was manually tuned based on the evaluation of the results against the validation set (10% of the original training set).

File	Selected value	Tested values
Batch size	32	32
Learning rate	0.001	0.01 & 0.001
Optimizer	Adam	SGD, Adagrad, RMSprop, Adam
Number of epochs	5	5

This time Adam was the best optimizer.

The learning rate 0.01 was an issue with the gradient descent as the accuracy stayed at the same level (around 10%) for a long time.

```
Epoch 1/5
1688/1688 [=====] - 204s 121ms/step - loss: 2.3063 - accuracy: 0.1004 - val_loss: 2.3035 - val_accu
racy: 0.1000
Epoch 2/5
1688/1688 [=====] - 208s 124ms/step - loss: 2.3038 - accuracy: 0.0997 - val_loss: 2.3034 - val_accu
racy: 0.1000
Epoch 3/5
1688/1688 [=====] - 201s 119ms/step - loss: 2.3039 - accuracy: 0.0990 - val_loss: 2.3033 - val_accu
racy: 0.1000
Epoch 4/5
1688/1688 [=====] - 191s 113ms/step - loss: 2.3040 - accuracy: 0.0978 - val_loss: 2.3035 - val_accu
racy: 0.1000
Epoch 5/5
1688/1688 [=====] - 192s 114ms/step - loss: 2.3040 - accuracy: 0.0989 - val_loss: 2.3034 - val_accu
racy: 0.1000
```

```
] : model2.optimizer.get_config()
```

```
{'name': 'Adam',
 'learning_rate': 0.01,
 'decay': 0.0,
 'beta_1': 0.9,
 'beta_2': 0.999,
 'epsilon': 1e-07,
 'amsgrad': False}
```

➔ With learning rate 0.001, the gradient descent found its way quickly.

Now, let's have a look to the architecture in 2 steps:

CNN part:

File	Filters number	Filter size	Stride / Max pooling	Padding	Activation
1 st conv layer	64	3x3	1*1	same	relu
Max pooling			2x2		
2 nd conv layer	128	3x3	1*1	same	relu
3 rd conv layer	128	3x3	1*1	same	relu
Max pooling			2x2		
2 nd conv layer	256	3x3	1*1	same	relu
Max pooling			2x2		
Flatten					

Fully connected layer part:

File	Selected value	Tested values
Activation function	Relu And finally softmax	Relu & softmax
Hidden layers	2	1, 2
Number of neurons per layer	128 and then 64	128, 64

Here is the final architecture of this neural net:

```

model2 = Sequential()

model2.add(Conv2D(64, (3, 3), strides = (1,1), padding='same', activation="relu", input_shape=X_train.shape[1:],))
model2.add(MaxPooling2D(pool_size=(2, 2)))

model2.add(Conv2D(128, (3, 3), strides = (1,1), padding='same', activation="relu"))
model2.add(Conv2D(128, (3, 3), strides = (1,1), padding='same', activation="relu"))
model2.add(MaxPooling2D(pool_size=(2, 2)))

model2.add(Conv2D(256, (3, 3), strides = (1,1), padding='same', activation="relu"))
model2.add(MaxPooling2D(pool_size=(2, 2)))

model2.add(Flatten())

model1.add(keras.layers.Dense(128, activation="relu"))
model2.add(Dropout(0.5))

model1.add(keras.layers.Dense(64, activation="relu"))
model2.add(Dropout(0.5))

model2.add(keras.layers.Dense(NUM_CLASSES, activation="softmax"))

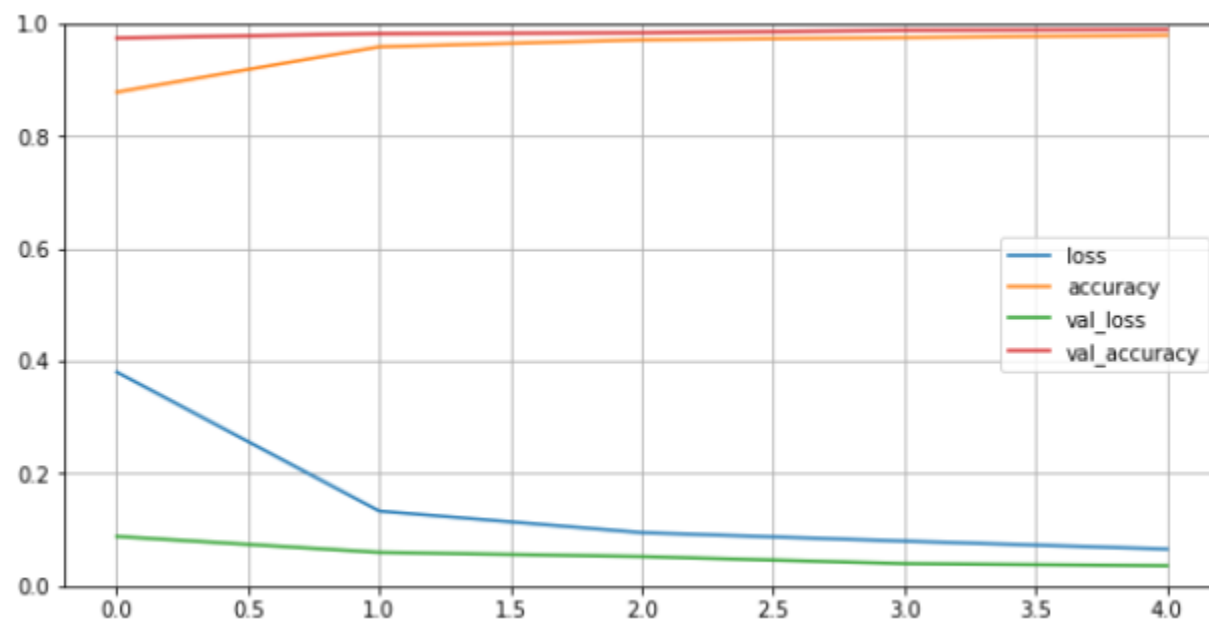
```

Layer (type)	Output Shape	Param #
conv2d_16 (Conv2D)	(None, 28, 28, 64)	640
max_pooling2d_11 (MaxPooling)	(None, 14, 14, 64)	0
conv2d_17 (Conv2D)	(None, 14, 14, 128)	73856
conv2d_18 (Conv2D)	(None, 14, 14, 128)	147584
max_pooling2d_12 (MaxPooling)	(None, 7, 7, 128)	0
conv2d_19 (Conv2D)	(None, 7, 7, 256)	295168
max_pooling2d_13 (MaxPooling)	(None, 3, 3, 256)	0
flatten_5 (Flatten)	(None, 2304)	0
dropout_8 (Dropout)	(None, 2304)	0
dropout_9 (Dropout)	(None, 2304)	0
dense_17 (Dense)	(None, 10)	23050
Total params: 540,298		
Trainable params: 540,298		
Non-trainable params: 0		

And then the results after 5 epochs.

```
Epoch 5/5
1688/1688 [=====] - 200s 118ms/step - loss: 0.0656 - accuracy: 0.9796 - val_loss: 0.0360 - val_accuracy: 0.9898
```

Let's plot the loss & accuracy for the training and the validation set:



Both accuracies and losses reached a plateau after 3 epochs.

Now, let's perform a test on the test set:

```
: model2.evaluate(X_test, y_test)
313/313 [=====] - 7s 22ms/step - loss: 0.1529 - accuracy: 0.9604
[0.15289919078350067, 0.9603999853134155]
```

Here are the summarized results for this model:

Type of accuracy	Score
Training Accuracy	97.96
Validation Accuracy	98.98
Test Accuracy	96.04

➔ The test accuracy is very good (around 96) compare to the previous NN (around 87).

CNN RESNET-34 NEURAL NETWORK

The 3rd model is a CNN ResNet-34.

This model was manually tuned based on the evaluation of the results against the validation set (10% of the original training set).

File	Selected value	Tested values
Batch size	32	32
Learning rate	0.01	0.01 & 0.001
Optimizer	Adam	Adam
Number of epochs	5	5

Here is the final architecture of this neural net (inspired by an example in Aurélien GERON book on Deep Learning):

```
model3 = keras.models.Sequential()
model3.add(keras.layers.Conv2D(64, 7, strides=2, input_shape=X_train.shape[1:],
padding="same", use_bias=False))
model3.add(keras.layers.BatchNormalization())
model3.add(keras.layers.Activation("relu"))
model3.add(keras.layers.MaxPool2D(pool_size=3, strides=2, padding="same"))
prev_filters = 64
for filters in [64] * 3 + [128] * 4 + [256] * 6 + [512] * 3:
    strides = 1 if filters == prev_filters else 2
    model3.add(ResidualUnit(filters, strides=strides))
    prev_filters = filters
model3.add(keras.layers.GlobalAvgPool2D())
model3.add(keras.layers.Flatten())
model3.add(keras.layers.Dense(10, activation="softmax"))
model3.summary()
```

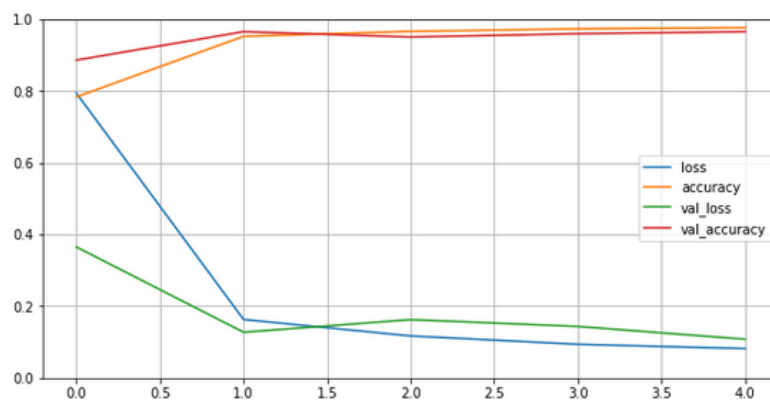
Layer (type)	Output Shape	Param #
conv2d_108 (Conv2D)	(None, 14, 14, 64)	3136
batch_normalization_108 (Batch Normalization)	(None, 14, 14, 64)	256
activation_3 (Activation)	(None, 14, 14, 64)	0
max_pooling2d_3 (MaxPooling2D)	(None, 7, 7, 64)	0
residual_unit_48 (ResidualUnit)	(None, 7, 7, 64)	74240
residual_unit_49 (ResidualUnit)	(None, 7, 7, 64)	74240
residual_unit_50 (ResidualUnit)	(None, 7, 7, 64)	74240
residual_unit_51 (ResidualUnit)	(None, 4, 4, 128)	230912
residual_unit_52 (ResidualUnit)	(None, 4, 4, 128)	295936
residual_unit_53 (ResidualUnit)	(None, 4, 4, 128)	295936
residual_unit_54 (ResidualUnit)	(None, 4, 4, 128)	295936
residual_unit_55 (ResidualUnit)	(None, 2, 2, 256)	920576
residual_unit_56 (ResidualUnit)	(None, 2, 2, 256)	1181696
residual_unit_57 (ResidualUnit)	(None, 2, 2, 256)	1181696
residual_unit_58 (ResidualUnit)	(None, 2, 2, 256)	1181696
residual_unit_59 (ResidualUnit)	(None, 2, 2, 256)	1181696
residual_unit_60 (ResidualUnit)	(None, 2, 2, 256)	1181696
residual_unit_61 (ResidualUnit)	(None, 1, 1, 512)	3676160
residual_unit_62 (ResidualUnit)	(None, 1, 1, 512)	4722688
residual_unit_63 (ResidualUnit)	(None, 1, 1, 512)	4722688
global_average_pooling2d_3 (Global Average Pooling2D)	(None, 512)	0
flatten_3 (Flatten)	(None, 512)	0
dense_3 (Dense)	(None, 10)	5130
Total params: 21,300,554		
Trainable params: 21,283,530		
Non-trainable params: 17,024		

This model was very long to train (more than 4 hours).

And then the results after 5 epochs.

```
Epoch 5/5
1688/1688 [=====] - 2621s 2s/step - loss: 0.0818 - accuracy: 0.9769 - val_loss: 0.1079 -
val_accuracy: 0.9657
```

Let's plot the loss & accuracy for the training and the validation set:



Both accuracies and losses reached a plateau after 1 epoch.

Now, let's perform a test on the test set:

```
] : model3.evaluate(X_test, y_test)
313/313 [=====] - 15s 48ms/step - loss: 0.2486 - accuracy: 0.9283
[0.24858123064041138, 0.9283000230789185]
```

Here are the summarized results for this model:

Type of accuracy	Score
Training Accuracy	97.69
Validation Accuracy	96.57
Test Accuracy	92.83

➔ The test accuracy is lower (around 93) compare to the previous CNN (around 96).

DEEP LEARNING MODEL RECOMMENDATION

➔ A paragraph explaining which of your Deep Learning models you recommend as a final model that best fits your needs in terms of accuracy or explainability.

Here are the results for all of the 3 models:

Model	Type of accuracy	Score
1st model: Fully connected NN	Training Accuracy	97.87
	Validation Accuracy	94.93
	Test Accuracy	87.99
2nd model: CNN	Training Accuracy	97.96
	Validation Accuracy	98.98
	Test Accuracy	96.04
3rd model: CNN ResNet-34	Training Accuracy	97.69
	Validation Accuracy	96.57
	Test Accuracy	92.83

➔ The best accuracy is given by the 2nd model, a CNN. He is better than the fully connected NN and also better than the CNN ResNet-34 that could have been better tuned.

KEY FINDINGS AND INSIGHTS

➔ Summary Key Findings and Insights, which walks your reader through the main findings of your modeling exercise.

As for its great cousin the MNIST dataset, it is quite easy to reach a good level of accuracy.

Thus, the data stakeholders might extend the project to all hiragana characters (there are 46 basic hiragana characters) as well as all katakana characters (2nd Japanese alphabet, 46 basic katakana characters).

As already mentioned earlier, this can also be extended to the 3rd type of written Japanese characters known as kanjis, this might be a more complex project as there are at least 3,000 kanjis in Japanese. But, if all of this is done, this could be used to translate old handwritten Japanese documents in another language. A recurrent neural network (RNN) might be added to understand whole sentence meaning (context).

SUGGESTIONS FOR NEXT STEPS IN ANALYZING THIS DATA

➔ *Suggestions for next steps in analyzing this data, which may include suggesting revisiting this model or adding specific data features to achieve a better model.*

I manually tuned the hyperparameters but I could have used a dedicated hyperparameters optimization framework such as KERAS tuner.

We could also have used data augmentation techniques, even if in that case, we already have a good number of entries (60,000 for the training set).

Another way to manage computer vision problem is to use transfer learning with a pre-trained neural network. I was looking for a transfer learning from VGG-19 but maybe it is not very adequate as this was trained on object images whereas here we are working with hand digits.

APPENDIX: CODE

The code for this project is available on GITHUB:

[https://github.com/Olivier-FONTAINE/IBM-Machine-Learning-professional-certificate/blob/main/05-DPL-Image%20classification-Hiragana%20\(KMNIST\).ipynb](https://github.com/Olivier-FONTAINE/IBM-Machine-Learning-professional-certificate/blob/main/05-DPL-Image%20classification-Hiragana%20(KMNIST).ipynb)