Step 1. Downloading data from Kaggle and importing libraries

```
from keras.preprocessing.image import ImageDataGenerator #Data augmentation and preprocessing
from keras.utils import to_categorical #For One-hot Encoding
from keras.optimizers import Adam, SGD, RMSprop #For Optimizing the Neural Network
from keras.callbacks import EarlyStopping
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import keras
from skimage import data, exposure, img_as_float

def show_image(arr):
    two_d = (np.reshape(arr, (28, 28)) * 255).astype(np.uint8)
    plt.imshow(two_d, interpolation='nearest')
    plt.show()

X_train = np.load('/kaggle/input/winter2020-mais-202/train_images.npy')[:45000]
y_train = pd.read_csv('/kaggle/input/winter2020-mais-202/train_labels.csv', index_col=0)[:45000]

X_test =  np.load('/kaggle/input/winter2020-mais-202/train_images.npy')[45000:50000]
y_test = pd.read_csv('/kaggle/input/winter2020-mais-202/train_labels.csv', index_col=0)[45000:50000]
```

.

```
X_train = X_train/255
X_test =X_test/255

from skimage import data, exposure, img_as_float
from skimage.filters import roberts, sobel, scharr, prewitt
X_train = X_train.reshape(-1,28,28)
X_test = X_test.reshape(-1,28,28)

X_train = exposure.adjust_gamma(X_train, 2)
X_test = exposure.adjust_gamma(X_test, 2)



X_train = X_train.reshape(-1,28,28,1)
y_train = keras.utils.to_categorical(np.array(y_train))

X_test = X_test.reshape(-1,28,28,1)
y_test = keras.utils.to_categorical(np.array(y_test))
```

We made the image to be dimmer to increase the accuracy.Then, we transfer the data into the NumPy matrix format in order to train it more easily. We split the data into two groups, 80% for training and 20% for validation. Moreover, we divide every element of the matrix by 255 so all the data is in the range of [0, 1]. Also, we reshape the images in 3 dimensions and convert the labels into the one-hot encoding.

```
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras import losses
from keras.regularizers import l2
from keras.callbacks import ReduceLROnPlateau
from keras.optimizers import RMSprop
from keras.regularizers import l2


model = Sequential()

model.add(keras.layers.BatchNormalization())
model.add(keras.layers.Conv2D(
    filters=32,
    kernel_size=(3,3),
    padding="same",
    activation="relu",
    input_shape = (28,28,1)
))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.Conv2D(
    filters=32,
    kernel_size=(3,3),
    padding="same",
    activation="relu"
))
,,
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.MaxPooling2D(pool_size=(2, 2)))
model.add(keras.layers.Dropout(0.5))
model.add(keras.layers.Conv2D(
    filters=64,
    kernel_size=(3,3),
    padding="same",
    activation="relu"
))
model.add(keras.layers.Conv2D(
    filters=64,
    kernel_size=(5,5),
    padding="same",
    activation="relu"
))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.MaxPooling2D(pool_size=(2, 2)))
model.add(keras.layers.Dropout(0.5))

model.add(keras.layers.Flatten())
model.add(keras.layers.BatchNormalization())

model.add(keras.layers.Dense(1024, activation='relu'))
model.add(keras.layers.Dense(10, activation='softmax',kernel_regularizer=l2(0.01)))
learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy',

                                            patience=3,
                                            verbose=1,
                                            factor=0.7,
                                            min_lr=0.00001)
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adam(lr=0.0012),
              metrics=['accuracy'])
```

```
model.fit(X_train, y_train,
          batch_size=64,
          epochs=50,
          verbose=1,
          validation_data=(X_test, y_test)
          ,callbacks=[learning_rate_reduction])
```

We used the Keras Sequential API to implement our model, which allows us to implement the model layer-by-layer. We use four filters and two pooling layer for one step of our model.

Then, to reduce overfitting, we use l2 regularization, drop, and batch normalization, and we employ call back to control the learning speed.

'relu' is the rectifier (activation function max(0,x). The rectifier activation function is used to add non linearity to the network.

The Flatten layer is use to convert the final feature maps into a one single 1D vector. It combines all the found local features of the previous convolutional layers.

```
test = np.load('/kaggle/input/winter2020-mais-202/test_images.npy')
test = test.reshape(-1,28,28)
test = test/255

test = exposure.adjust_gamma(test, 2)
test = test.reshape(-1,28,28,1)
predict = model.predict(test,batch_size=256,verbose=1)

predict = np.argmax(predict,axis=1)

df_test = pd.read_csv('/kaggle/input/winter2020-mais-202/sample_submission.csv')
df_test['label'] = predict
df_test.to_csv('submission.csv', index=False)
```

Finally, we used our model to predict the test set. We also make the test set dimmer to match what we did to our train set.

Results:

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
batch_normalization_6 (Batch (None, 28, 28, 1)         4
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 28, 28, 32)        320
_________________________________________________________________
batch_normalization_7 (Batch (None, 28, 28, 32)        128
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 28, 28, 32)        9248
_________________________________________________________________
batch_normalization_8 (Batch (None, 28, 28, 32)        128
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
dropout_3 (Dropout)          (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 14, 14, 64)        18496
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 14, 14, 64)        102464
_________________________________________________________________
batch_normalization_9 (Batch (None, 14, 14, 64)        256
```

```
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
dropout_4 (Dropout)          (None, 7, 7, 64)          0
_________________________________________________________________
flatten_2 (Flatten)          (None, 3136)              0
_________________________________________________________________
batch_normalization_10 (Batc (None, 3136)              12544
_________________________________________________________________
dense_3 (Dense)              (None, 1024)              3212288
_________________________________________________________________
dense_4 (Dense)              (None, 10)                10250
=================================================================
Total params: 3,366,126
Trainable params: 3,359,596
Non-trainable params: 6,530
```

Our result accuracy is around 90.6%

Challenge: At first, we forgot to divide test set by 255, which leads to very low submission point and high validation accuracy. Then,when adjusting our model at first, we find it hard to reach 90% accuracy and we faced overfitting. Then, we add callback to control the learning speed, which results in lower loss and higher accuracy. To minimize the effect of overfitting, we use more drop out and applies batch normalization.

Conclusion: We learned how to imply the CNN model for image classification. Through designing process, we learned different way to improve our models, and we saw how they affect the final result of our model. Some of them, like callback and batch normalization are very efficient, while some have little effect.

Individual Contribution: We design the model together. Zhiwei Li focused on the improvement of the parameter and new methods, and Ruixi Jiang focused on \dealing with original data and finding CNN image classification examples online.