High Precise Kannada Digit Recogniser with CNNs and "Swish" Activation Function Approach

0.1 Abstract

Kannada digits are different than standard arabic digits. Over 50 million people that liver India uses those digits in their daily life. Kannada MNIST dataset is prepared like MINIST but the detection of numbers are harder than standard arabic digits. CNN Based high precise MNIST like Kannada digit recognizer is created for a competition is organized by kaggle. With this solution, The competition is completed in top 4 percentage.

1 Introduction

The goal of this competition and project is to provide a simple extension to the classic MNIST competition we're all familiar with. Instead of using Arabic numerals, it uses a recently-released dataset of Kannada digits. Kannada is a language spoken predominantly by people of Karnataka in southwestern India. The language has roughly 45 million native speakers and is written using the Kannada script.

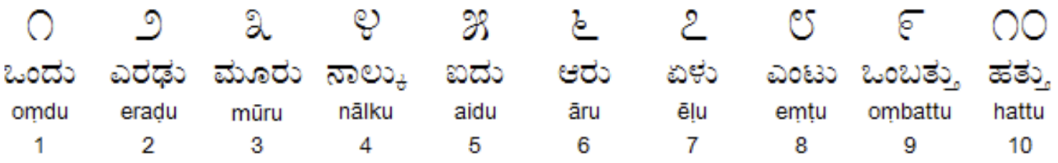

Figure 1: Kannada Digits

2 Development Environment

Kaggle Cloud Computing, GPU Based Tensorflow 2.0

3 Architecture

CNN architecture is used to build a model. To increase accuracy, different activation function that generally used with CNNs like ReLu, Sigmoid etc. The function is named swish is not

generally described on Keras is used. Swish performs slightly better accuracy than sigmoid, ReLu or LeakyReLu. On the other hand, I comes with a computation cost. Because the models that trained with swish works slower than other models.

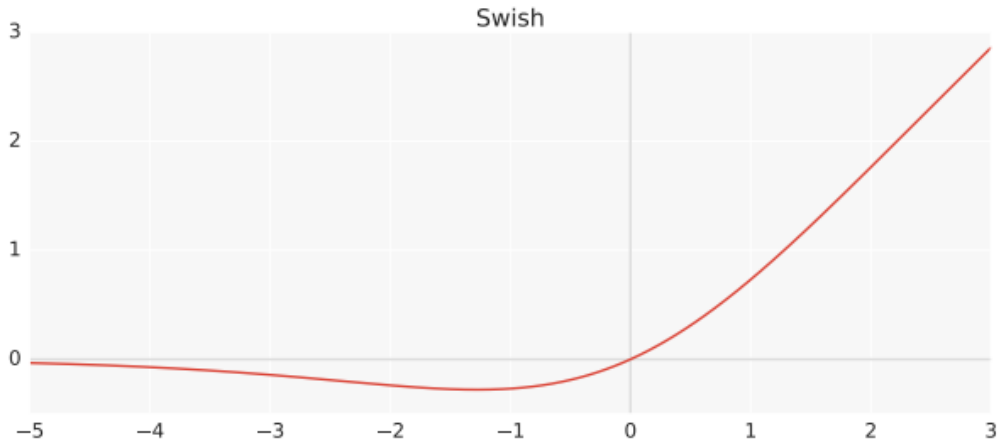

Figure 2: Swish Activation function

Quite complex solution and architecture for digit recognizer. But It increased the accuracy, thanks to this "heavy" architecture, my solution comes top 4percentage of the competition.

Optimizers, In this project RMSProp is used with optimizer function. Due to many sources, Adam optimizer offers better accuracy for solutions but In my solution RMSProp behave better. On the other hand, earlystopping and ReduceLROnPlateau features of keras. They decrease training time and prevent overfitting.

The another feature of my project is adapting learning rate. Optimal learning rate is found for this solution and used.

The least but not the least feature and requirement of the solution is data enhancement. Before fedding data into the system, it needs to be enhance. Prevent overfitting, some images need to be changed, It means we create new images with some manipulating operations (rotate, contrast changes, filters etc.)

```
datagen = ImageDataGenerator(
    rotation_range=8,
    width_shift_range=0.24,
    height_shift_range=0.24,
    shear_range = 0.1,
    zoom_range = 0.24,
    horizontal_flip = False)
```

The model layers look like:

```
get_custom_objects().update({'swish': Activation(swish )})
model.add(Conv2D(64, kernel_size= (3,3), input_shape=(28, 28, 1),padding='same'))

model.add(BatchNormalization(momentum=0.5, epsilon=1e-5, gamma_initializer="uniform"))
model.add(LeakyReLU(alpha=0.1))
model.add(Conv2D(64, kernel_size=(3,3), padding='same', activation='swish'))
model.add(BatchNormalization(momentum=0.1, epsilon=1e-5, gamma_initializer="uniform"))
model.add(LeakyReLU(alpha=0.1))

model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.35))

model.add(Conv2D(128, kernel_size =(3,3),padding='same', activation='swish'))
model.add(BatchNormalization(momentum=0.2, epsilon=1e-5, gamma_initializer="uniform"))
model.add(LeakyReLU(alpha=0.1))
model.add(BatchNormalization(momentum=0.1, epsilon=1e-5, gamma_initializer="uniform"))
model.add(LeakyReLU(alpha=0.1))
model.add(Conv2D(128,(3,3), padding='same', activation='swish' ))
model.add(BatchNormalization(momentum=0.1, epsilon=1e-5, gamma_initializer="uniform"))
model.add(LeakyReLU(alpha=0.1))

model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.35))

model.add(Conv2D(256, kernel_size = (3,3), padding='same', activation='swish'))
model.add(BatchNormalization(momentum=0.2, epsilon=1e-5, gamma_initializer="uniform"))
model.add(LeakyReLU(alpha=0.1))
model.add(Conv2D(256, kernel_size= (3,3) ,padding='same', activation='swish'))
model.add(BatchNormalization(momentum=0.1, epsilon=1e-5, gamma_initializer="uniform"))
model.add(LeakyReLU(alpha=0.1))

model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.35))
model.add(Flatten())
model.add(Dense(256))
model.add(LeakyReLU(alpha=0.1))
model.add(BatchNormalization())
model.add(Dense(10, activation='softmax'))
model.summary()
```

4 Results

Results of solution is very good and satisfied. I get Over 0.99 + accuracy every time.

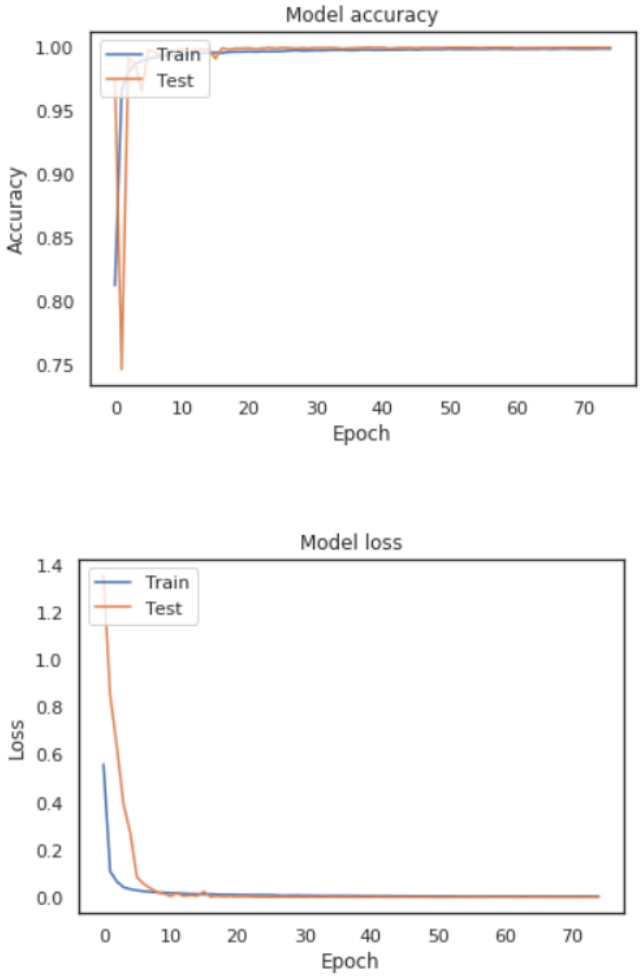

Figure 3: Accuracy and Loss Graphs

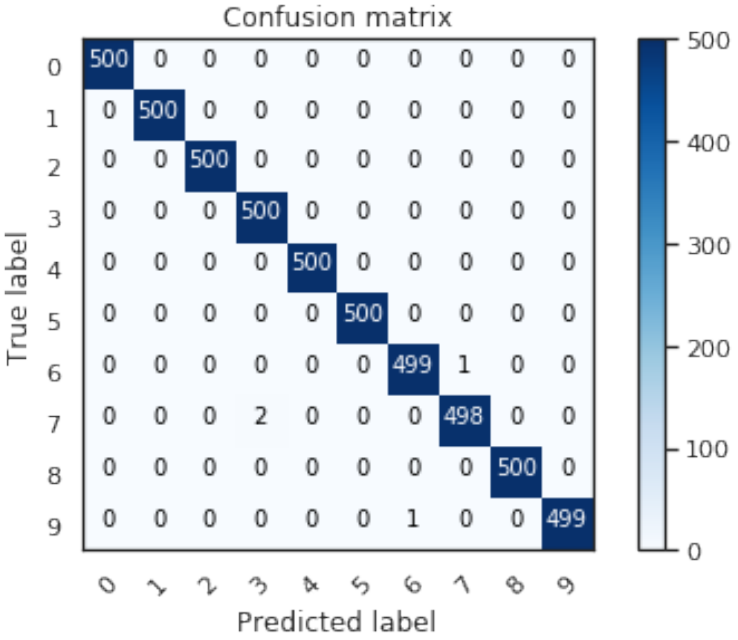

Figure 4: Accuracy and Loss Graphs

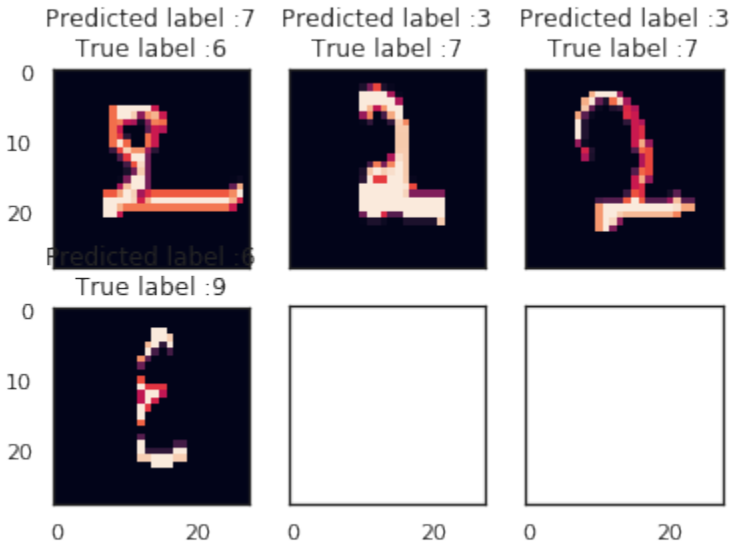

Figure 5: Found False/Positives and True/Negatives