BITS-F464 MACHINE LEARNING

Project Final Report

Group Number:ML24

Members and ID Numbers:

1. Yasovar Tammareddy 2019AAPS0226H
2. Yarramsetty Sanjeeva Sai Preetham 2019A3PS0485H
3. Praneet Surabhi 2019A7PS0060H
4. A P Karthikeya 2019AAPS0276H
5. Anjan Neelisetty 2019A8PS0367H

Github Link :https://github.com/Psylence0609/Telugu-Character-Recognition

The Dataset:

The dataset contains 1200 images each image of size 28x28. It contains the first six vowels of the Telugu Alphabet. The images are first converted into binary type and then split into train,test and valid sets in the ratio of 81:10:9 for any Deep Learning method that we may use.To compute the efficiency, we used a confusion matrix to find all the false positives and true positives on the training dataset.

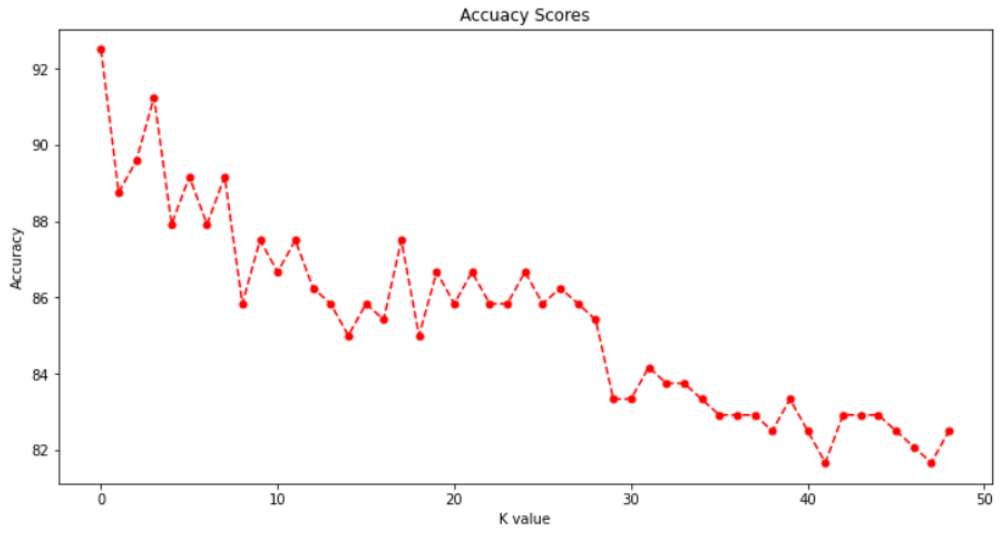
Implementation of Research Paper

Paper: P. N. Sastry, T. R. V. Lakshmi, N. V. K. Rao, T. V. Rajinikanth and A. Wahab, "Telugu Handwritten Character Recognition Using Zoning Features"

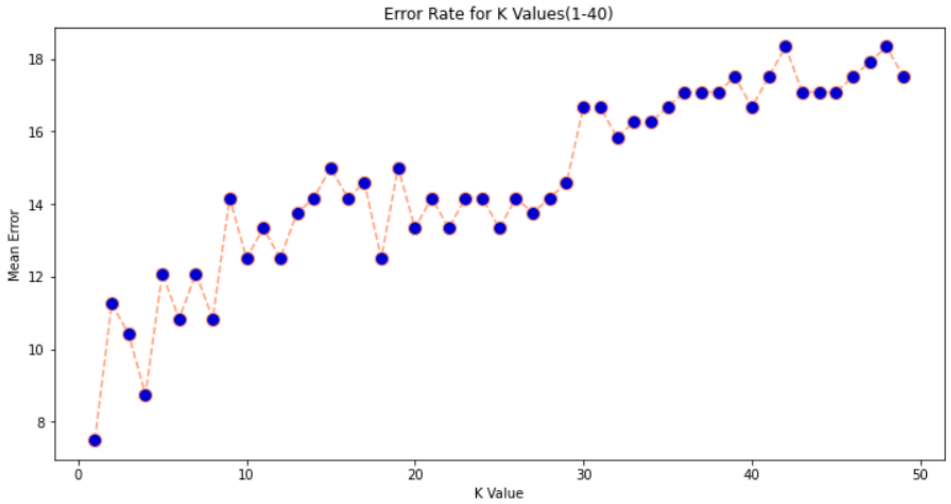
The paper uses a Nearest Neighbour classifier to identify and classify the results. The procedure is as follows:

1. Load and read all the images.
2. Convert the images into binary type using a threshold 0.7.
3. Divide the image into 49 zones of size 4x4.
4. The feature vector is the sum of all pixel intensities in the zone.
5. Reshape the features into a size of 49x1
6. Repeat the above procedure for all the images.
7. Apply KNN on the feature vectors.(Finding the euclidean distance and taking the mod of K nearest neighbours.)

Accuracy Vs K Value:



Mean Error Vs K Value:



So, from the given data about the varying accuracy with K-value we can come to the conclusion that the KNN gives the best accuracy when the K-value is 1 and the accuracy that we obtain is 92.5%

```
:  max_acc=max(accuracy)
   max_acc_k=accuracy.index(max_acc)+1
   print(max_acc_k)
   print(max_acc)
```

```
1
92.5
```

The dataset that we have used is different from the one used in the research paper. We could not find the exact dataset that was used for the paper. The website which provided the dataset has been shut down. Due to lack of resources we had to implement based on the above smaller dataset which can be downloaded from the Kaggle website.

The original paper had only 78% percent accuracy and hence our model is way better.

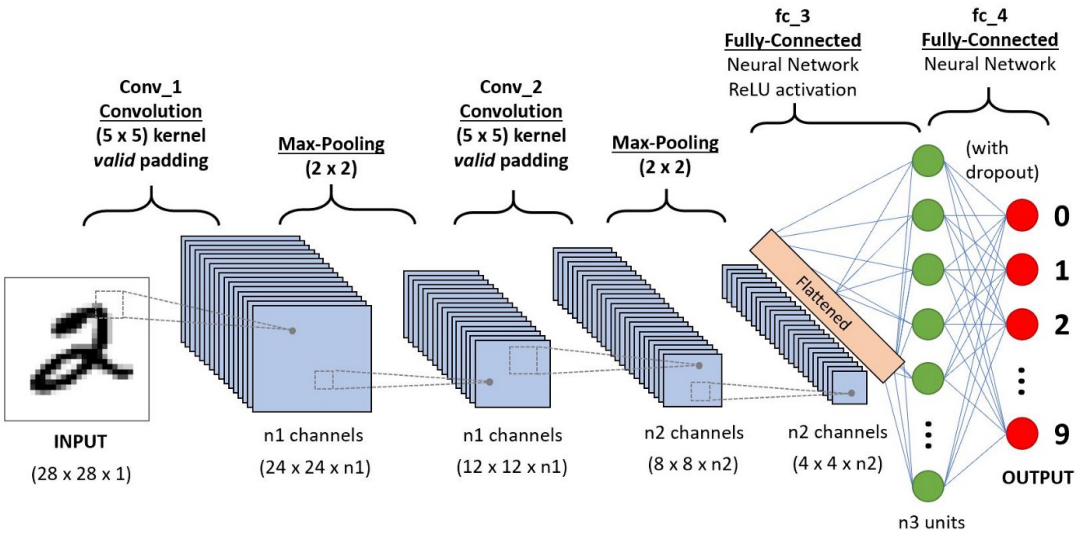
Improving the results with CNNs:

Convolutional Neural Networks:

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning system that can take an input picture, assign relevance (learnable weights and biases) to various aspects/objects in the image, and distinguish between them. When compared to other classification methods, the amount of pre-processing required by a ConvNet is significantly less. While basic approaches need hand-engineering of filters, ConvNets can learn these filters/characteristics with enough training.

Each layer consists of a convolution layer followed by a pooling layer. A convolutional layer performs a special operation using a kernel of an arbitrary size and helps in capturing spatial and temporal dependencies. A pooling layer uses another filter and it has 2 types. A max pooling layer takes the maximum value in the window while an average pooling layer takes the mean of the values in the window.

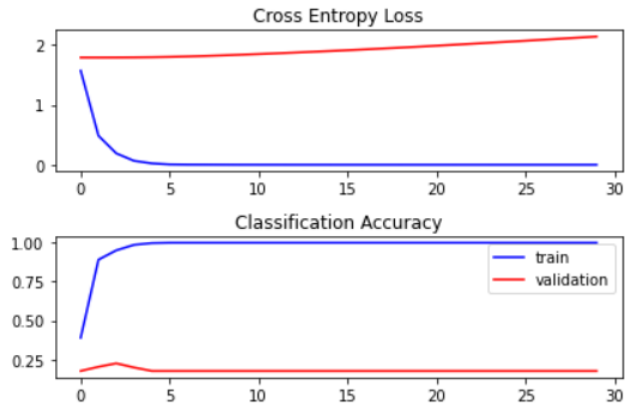
After a few CNN layers we flatten the image and are connected to a feed forward network and the images are classified using a softmax layer.



1. Model 1:

```
model = Sequential()
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu', padding='same', input_shape=(28,28,1)))
model.add(BatchNormalization())
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, kernel_size=(3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(256, kernel_size=(3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(256, kernel_size=(3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(1024, activation='relu'))
model.add(Dense(512, activation='relu'))
model.add(Dense(6, activation='softmax'))

optim = SGD(learning_rate=0.01, momentum=0.9)
model.compile(optimizer=optim, loss='categorical_crossentropy', metrics=['accuracy'])
return model
```

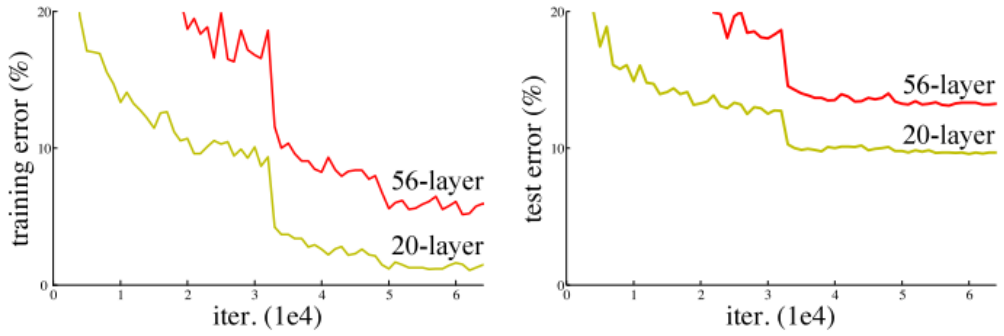
```
Test Accuracy: 12.666666507720947 %
```



Tackling overfitting:

The above model has high training accuracy and low testing accuracy. Which means that the model has high variance. There is a bias-variance tradeoff in any model.

Bias occurs when the model is oversimplified. It leads to underfitting of the dataset. Hence increasing the complexity of our model generally solves the problem. But increasing the complexity causes overfitting and high variance. Hence there is a tradeoff.



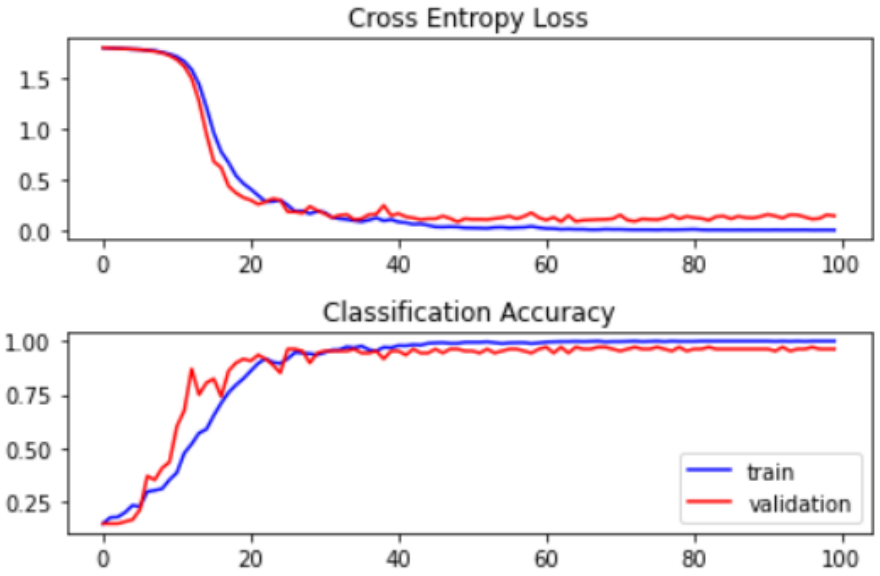
2. Model 2:

```
model = Sequential()
image_size = 28
model.add(Conv2D(32, (5, 5), activation='relu', padding='same', input_shape=(image_size, image_size, 1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())

model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(6, activation='softmax'))
```

It has lower complexity than model 1 in terms of layers and filters. It has 4 convolution layers(32,32,64,128 filters) each followed by a max pooling layer. Then the image is flattened and connected to a fully connected layer with a dropout=0.5. Dropout also tackles overfitting by reducing the dependency on a single node(or factor). It basically drops different units and makes sure that no units are co-dependent.

For the results:

```
Test Accuracy: 97.50000238418579 %
```



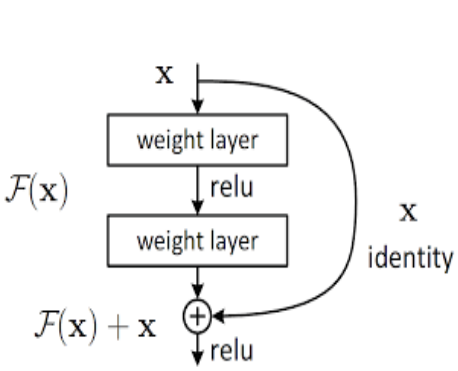
Improving the results with a ResNet:

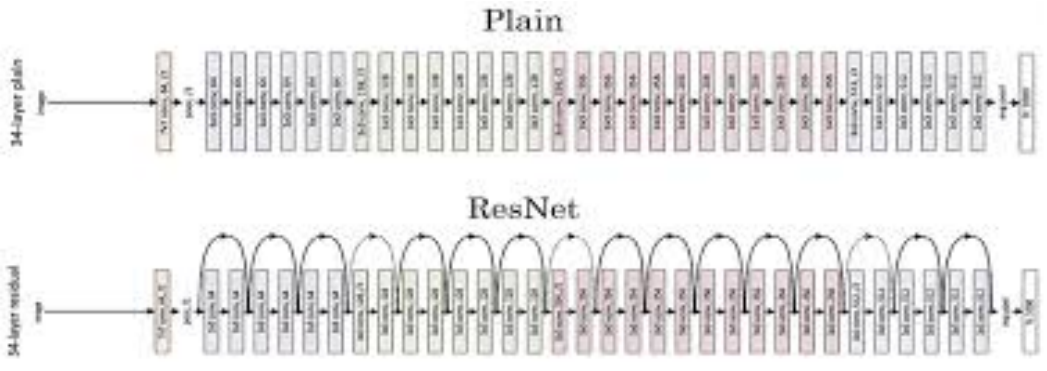
ResNet, a new architecture presented by Microsoft Research in 2015, established a new architecture called Residual Network.

Residual Block:

This design introduces the notion of the Residual Network to overcome the problem of the vanishing/exploding gradient. We employ a method called skip connections in this network. The skip connection bypasses a few stages of training and links straight to the output.Instead of allowing layers to learn the underlying mapping, we let the network to fit the residual mapping.

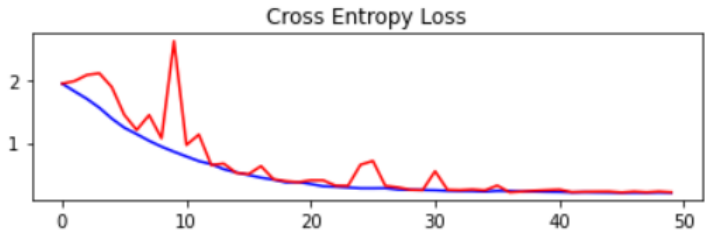
The benefit of including this sort of skip connection is that any layer that degrades architectural performance will be bypassed by regularisation. As a consequence, very deep neural networks may be trained without the issues caused by vanishing/exploding gradients. On the CIFAR-10 dataset, the authors of the research tested with 100-1000 layers.

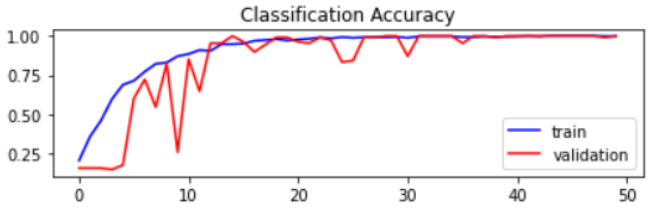


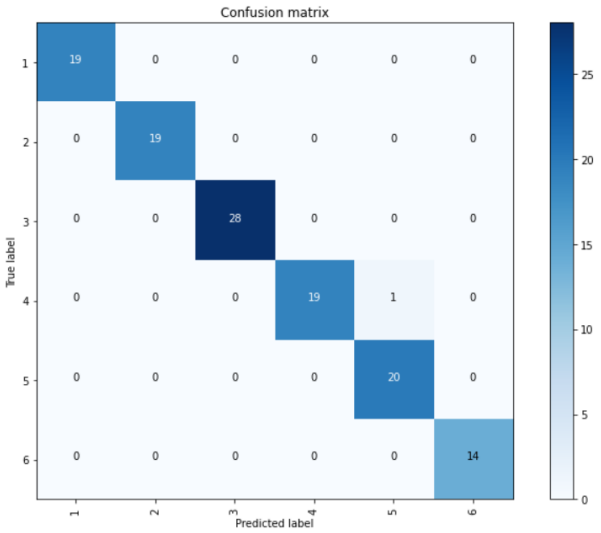


The results are as follows:

Training Accuracy : 99.78 % Validation accuracy : 100% Test Accuracy: 99.166%








```
Test Accuracy: 99.16666746139526 %
```

Findings & Accomplishments:

1. We have found that CNN performs better than KNN which is the published method.
2. Increasing the layers causes overfitting of data and decreases the test accuracy.
3. The best CNN model gave 97.5% accuracy on the test dataset and is more accurate than the KNN classifier.
4. ResNet helps in regularizing and tackling overfitting by implementing a deeper network without any tradeoff between complexity and accuracy.
5. ResNet gave nearly 99 % accuracy and the best result.