ECSE 551 - Mini-Project 3

Anonymous Author(s)
Affiliation
Address
email

Abstract

The purpose of this project was the implementation of a 10 classes image classifier using Convolutional Neural Networks (CNN) and other models. The dataset is made of 60000 labeled images containing one Japanese number and other characters from the EMNIST dataset. After iterating over different architectures, the best performing model was the VGG 5 achieving a maximum accuracy of 89.9 %.

Contents

1 Introduction **2**

2 Dataset **2**

2.1 Data Analysis 2

2.2 Pre-processing 3

3 Proposed Approach **3**

3.1 ConvNet 3

3.2 ResNet 4

3.3 VGG16 4

3.4 VGG19 4

3.5 VGG5 and VGG5 Custom 4

3.6 Training 5

4 Results **5**

5 Discussion and conclusion **6**

6 Statement of Contributions **6**

Submitted to 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Do not distribute.

1 Introduction

The objective of this project involved deploying a 10-class image classifier utilizing Convolutional Neural Networks (CNN) and alternative models, as previously indicated. The dataset we are classifying is the EMNIST data set, which is an extended version of the reknown MNIST Dataset. Each image contains at least one japanese number in addition to other characters. The images are all labeled with one digit from 0 to 9 indicating their belonging to one of 10 possible classes.

It is worth noting that the dataset we have received is of a considerable size (60 000 *image* samples). Therefore, in addition to accuracy concerns, we had to take into consideration the computing needed to train our model. This is why most of the models we have implemented were trained in the Google Colab environment, using the cuda parallel computing platform to accelerate the training process. Even then, most of our models had to be trained for long periods of time to obtain satisfying results.

For this reason, we were unable to perform 10-fold-cross validation for all our models, as we had done in the previous projects. However, other methods to prevent over fitting were implemented (such as L1 regularization for example, or the addition of dropout in our network).

We made sure all models imported were not pre-tained.

2 Dataset

2.1 Data Analysis

The training set is composed of 60 000 samples that belong to one of 10 classes, indicated by a single digit (0-9). Each image is in a grayscale format, and has a size of 28 x 28 pixels. The dataset is evenly distributed with about 6000 instances of each class. A sample image from the dataset was selected for illustrative purposes :

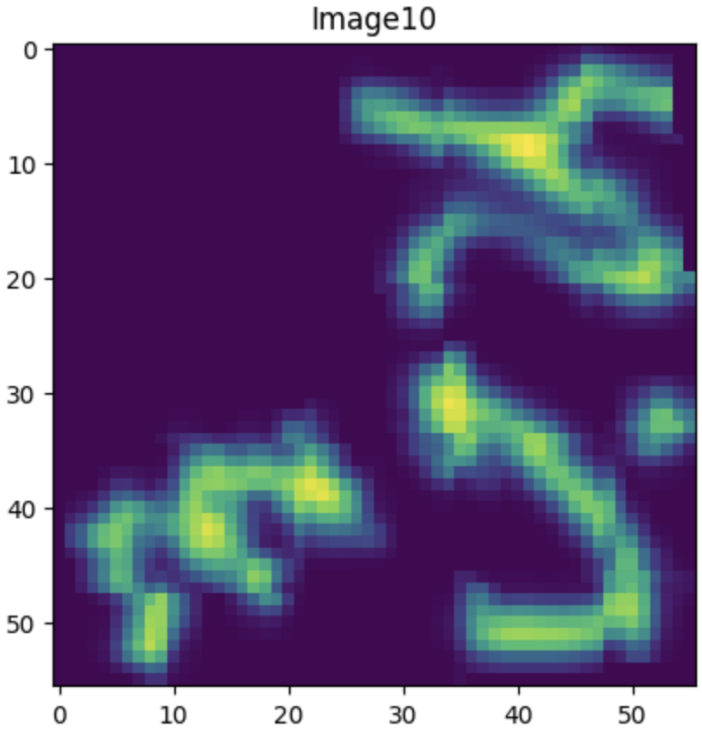

Figure 1: Sample image from EMNIST

Below is a histogram depicting the distribution of classes in our dataset:

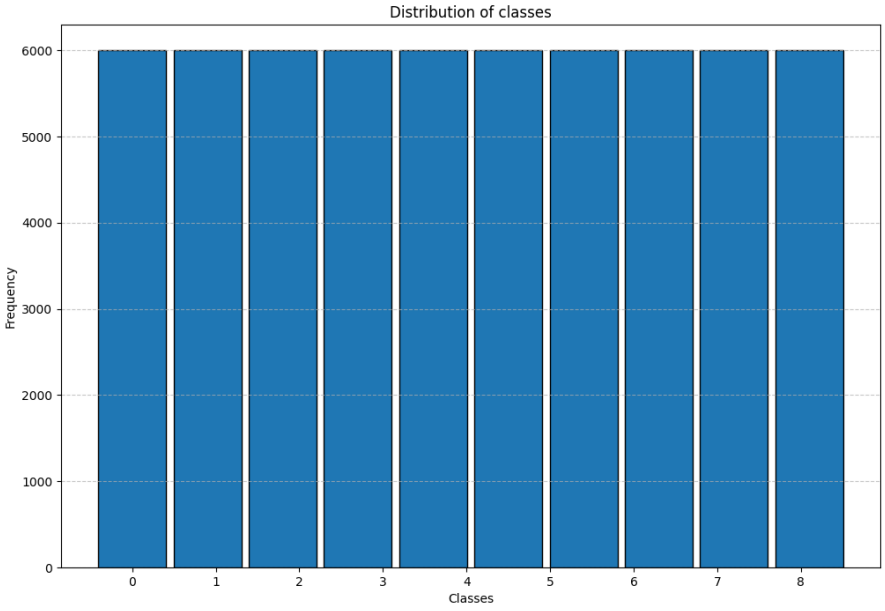

Figure 2: Distribution of classes in our dataset

2.2 Pre-processing

In order to reduce the risk of overfitting, the following transformations were done on the image samples : random horizontal and vertical flipping, random rotation and resizing. The resizing was implemented to make sure all image samples were of a homogeneous size and to have a better visualisation of our images. The rest of the augmentations were used to increase the diversity of the training set and make our model more generalized.

3 Proposed Approach

We first decided to implement several architectures and assess how they were performing. Then, we selected the most promising models for further training and a more in depth fine tuning. The input channels (designed for a 28 x 28 pixel, grayscale image) and the output (with a last layer of dimension 10 for a 10 class-classifier) are naturally shared amongst all models.

3.1 ConvNet

A basic CNN was implemented. Below is a graph showing the architecture of our model:

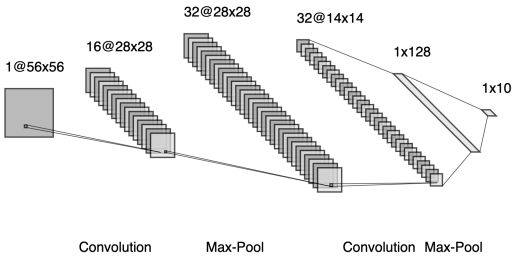

Figure 3: CNN Architecture

It consists of two Convolution - Max Pool layers followed by 2 fully connected layers.

3.2 ResNet

We implemented custom Residual Neural Network (ResNet) model. They're different than ConvNet in the sense that all layers are connected to each other. The architecture we have implemented is composed of an initial convolutional layer, followed by 2 sets of residual blocks each containing 64 channels and 128 channels respectively, and a stride of 2 for downsampling. The architecture is then completed with global average pooling, flattening, and a fully connected layer with 10 output neurons, representing the 10 output classes.

3.3 VGG16

We have selected the well known VGG16 architecture to be implemented. A brief description of the architecture of VGG16 is available below: As one can see from the image above, VGG16 is

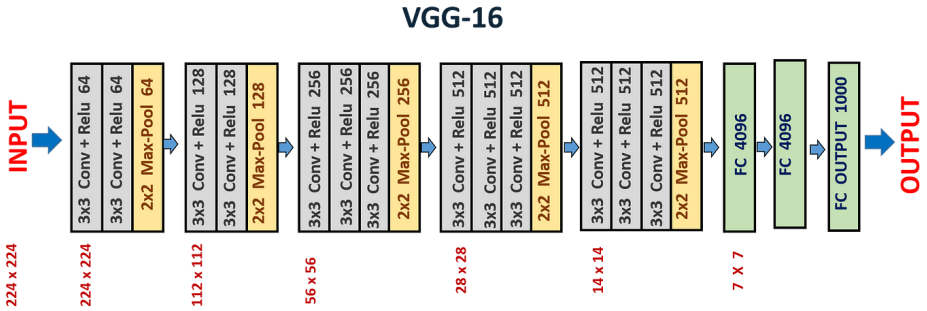

Figure 4: VGG16 architecture

composed of 13 convolutional layers, 5 max pooling layers, and 3 dense layers.

3.4 VGG19

VGG16 and VGG19 have a very similar architecture, the only difference between both architectures being that VGG19 has 3 more convolutional layers than VGG16. Both were implemented to highlight the effect of a deeper network for our classification task.

3.5 VGG5 and VGG5 Custom

After experimenting deeper architectures, we have decided to also implement more shallow architectures. The one we have selected is VGG5, which consist of 10 convolutional layers and 3 fully

connected layers. A custom version of VGG5 was also implemented with a different architecture. Both have the same depth but the regular one has a better generalization error and is more efficient.

3.6 Training

Due to the large computation requirements needed to train our models, the implementation of a nested K fold cross validation was simply too time consuming. Instead, we have decided to split the original dataset into 80% for training, and 20% for validation, resulting in 48000 training samples and 12000 validation samples. The optimizer selected was either Stochastic Gradient Descent (SGD) with a learning rate of 0.001 or Adam with the same rate. A learning rate scheduler was also added for some models. For the custom ResNet as well as CNN, training was done on the entirety of the training set. The batch chosen was 32 samples, as it is a power of 2 and was shown to provide the best results. All models were trained for at least 10 epochs so that we could highlight the most promising models. Models were trained from 10 to 30 epochs. Xavier/Glorot initialization was added and applied to ResNet and ConvNet models.

4 Results

A summary of our results can be found in the table below.

Table 1: Classifier Model Performance

Classifier	Model Accuracy (%)	Runtime (s)
VGG16	77.9	923.6
VGG19	56.7	1523.7
VGG5 Custom	82.2	**307.9**
VGG5	**89.9**	4311
SpinalVGG	86.6	615.5
CustomResNet	78.3	365.6
DenseNet	78.6	2216.8
CNN	73.4	300.3

As we can see, the best accuracy was achieved with VGG, while the lowest runtime execution was obtained with VGG 5 Custom. The best compromise between accuracy and runtime would be VGG 5 Custom since it achieved an accuracy of 82.2 % . Below is a graph depicting accuracy evolution through epochs by our custom ConvNet:

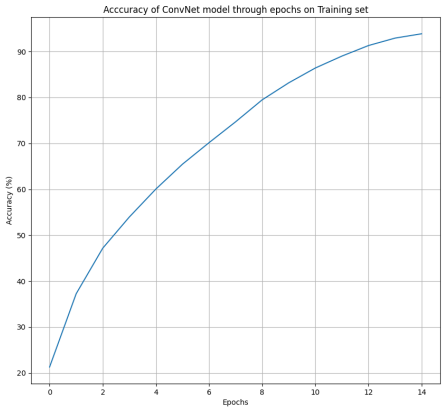

Figure 5: Acccuracy of ConvNet model through epochs on Training set

We can see that the accuracy converges logarithmically. Even though no validation set was made for this model, we can assume validation accuracy follows. This model was however not deep enough to achieve higher accuracy.

5 Discussion and conclusion

In conclusion, our goal was successfully achieved. Our best classifier turned out to be VGG5 with an accuracy of 89.9 % and a total runtime of 4311 s. Higher accuracy could be achieved with bigger computational ressources through fine-tuning and larger scale training. We tried to implement more complex models such as VGG19 and VGG16 but their architectures were too deep and would lead to overfitting and higher generalization error. We sought to fix these issues through nested K-fold cross validation and L1 regularization ; 1st method had a too large computational cost and L1 regularization did no effect. Dropouts were added but had a slight effect too.

6 Statement of Contributions

Aymen handled most of the pre-processing, as well as the implementation of the ConvNet, ResNet and Dense model. Hamza handled the implementation of the VGG models, and their fine tuning. Rest of the work was evenly shared.

References

VGG

Densely Connected Convolutional Networks