Image Classification on CIFAR-100 dataset using C-NN

Dimple Bapna

***Abstract*— Convolution Neural Networks have become state of the art method for image classification and recognition tasks over the last couple of years. Now they even surpass humans on many of the image classification datasets. The Convolutional Neural Networks (CNNs), in domains like computer vision, mostly reduced the need for handcrafted features due to its ability to learn the problem-specific features from the raw input data. Convolutional Neural Networks (CNNs) dominate various computer vision tasks since Alex Krizhevsky showed that they can be trained effectively and reduced the top-5 error from 26.2% to 15.3 % on the ImageNet large scale visual recognition challenge. Many aspects of CNNs are examined in various publications, but literature about the analysis and construction of neural network architectures is rare. This work is one step to close this gap. A comprehensive overview over existing techniques for CNN analysis and topology construction is provided. Results are confirmed and quantified for CIFAR-100.**

This paper proposes a method to recognize and classify images by only using CNNs. Experiments have been performed with CNN architectures having different depths and more focus is given on improving the accuracy of the model using various optimization techniques like hyperparameter tuning, regularization, normalization, image augmentation etc. After using some of these techniques and adjusting the learning rate according to the number of epochs, an accuracy of 48.39% was obtained on the model consisting of only 3 Fully Connected layers and 4 Convolution layers.

I. INTRODUCTION

Computer vision consists of different problems such as image classification, localization, segmentation and object detection. Among those, image classification can be considered as the fundamental problem and forms the basis for other computer vision problems. Until '90s only traditional machine learning approaches were used to classify image. But the accuracy and scope of the classification task were bounded by several challenges such as hand-crafted feature extraction process etc. In recent years, the deep neural network (DNN), also entitled as deep learning, finds complex formation in large data sets using the backpropagation algorithm. Among DNNs, convolutional neural network has demonstrated excellent achievement in problems of computer vision, especially in image classification.

Image classification/recognition is used in healthcare, security, education, autonomous vehicles etc. Correct categorization is vital. Image recognition, being a supervised classification problem, involves the use of training data that are considered representatives of each class to be classified. The project addresses this by using Deep Convolution Neural Networks to classify the image into one of the 100 categories and achieving the highest possible accuracy leveraging the CIFAR-100 Dataset.

However, to achieve a higher accuracy it is important to optimize the Convolution Neural Network. In this project, I tried to increase the number of layers to get better predictions but failed due to the problem of vanishing gradient. I have used several other strategies which employ number of layers, normalizing the data, choosing an activation function, learning rate schedules, batch normalization, dropout after convolution layers as well as fully connected layers, and have recorded the loss and accuracy for all the combinations of these strategies.

Finally, a convolution neural network with 4 convolution layers and 3 fully connected layers, along with an 'ELU' activation function and batch normalization was selected that achieved an accuracy of around 48% on CIFAR-100 dataset. It has a total of 100 classes, and each class contains 600 images (32 * 32 dimension)- 500 training images and 100 testing images. Since we have only 500 images per class to train our model, achieving an accuracy higher than this requires the use of other networks like ResNets, VGGs etc., and high computing power (preferably a GPU). This is the highest accuracy ever achieved using the traditional CNNs and a CPU with 8GB RAM.

II. BACKGROUND

A. *Fast and accurate deep network learning by exponential linear units (**ELUs**)*

Currently the most popular activation function for neural networks is ReLU given as follows:

Function	Derivative
$R(z) = \begin{cases} z & z > 0 \\ 0 & z <= 0 \end{cases}$	$R'(z) = \begin{cases} 1 & z > 0 \\ 0 & z < 0 \end{cases}$

Fig. 1. Forecasting Graph for Item 1108569.

The main advantage of ReLU is that they alleviate the vanishing gradient problem. However, they are always non-negative and so they have a mean activation much larger than zero; and hence acts as a bias for the next layer.

However, ELUs have negative value which allows them to push the mean activation near to zero, and this speed up the learning process by bringing the normal gradient closer to the unit natural gradient because of a reduced shift effect.

Function	Derivative
$R(z) = \begin{cases} z & z > 0 \\ \alpha.(e^z - 1) & z <= 0 \end{cases}$	$R'(z) = \begin{cases} 1 & z > 0 \\ \alpha.e^z & z < 0 \end{cases}$

Fig. 2. Forecasting Graph for Item 1108569.

In fact, it is proven that ELUs not only lead to faster learning, but also significantly better generalization than ReLUs.

B. *Densely Connected Convolution Networks*

Recent work has shown that convolutional networks can be substantially deeper, more accurate, and efficient to train if they contain shorter connections between layers close to the input and those close to the output. Existing research on Dense Convolutional Network (DenseNet), which connects each layer, have several compelling advantages: they alleviate the vanishing-gradient problem, strengthening feature propagation, encourage feature reuse, and substantially reduce the number of parameters using CIFAR-10, CIFAR-100, SVHN, and ImageNet as benchmarking datasets.

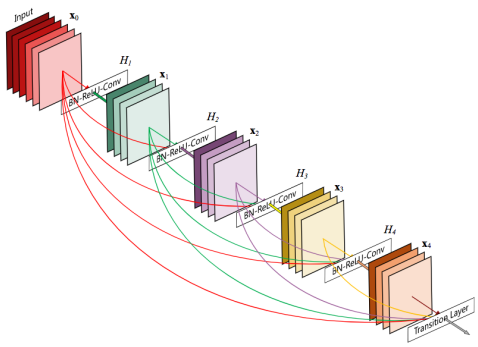


Fig. 3. DenseNet

III. APPROACH

One of the most difficult and the most important task while designing a deep convolution neural network is to optimize the model by tuning it's hyperparameters. Various approaches have been followed here- which reveals the importance of tuning the hyperparameters and use of regularization to avoid overfitting. In this project I used the following approach to create and best optimize my model.

A. *Data Preparation and preprocessing*

Loaded the dataset directly from TensorFlow library; and divided it into training set consisting of 50,000 images and testing set consisting of 10,000 images. Explored the dataset, and visualized random images.

Fig. 4. CIFAR-100.

Since, I am using CNN for this project, the original format is inappropriate. In order to feed image data into a CNN model, the dimension of the input tensor should be in the format required by TensorFlow's tensor object which is *(width*height*num_channel)*. Since, these are colored 32*32 pixel images, the input format will be *(3,32,3)*.
So, data was reshaped accordingly. After this, I normalized the data to transform the original image that ranges from 0 to 255 to an image that ranges from 0 to 1, to avoid the possible chances of exploding/vanishing gradient problems when back propagation is performed to optimize the network.

B. *Create a Model*

Once the input is ready, the next step is to create a model. The entire model consists of 8 layers in total. In addition to the layers, various techniques were applied to avoid overfitting and improve the overall performance of the model which is as follows:

Convolution with 32 different filters in size of (3, 3)
kernel_regualarizer l2 with weight decay of 0.005
ELU activation function
Batch Normalization
MaxPooling by 2

Convolution with 64 different filters in size of (3, 3)
kernel_regualarizer l2 with weight decay of 0.005
ELU activation function
Batch Normalization
MaxPooling by 2

Convolution with 128 different filters in size of (3, 3)
kernel_regualarizer l2 with weight decay of 0.001
ELU activation function
Batch Normalization
MaxPooling by 2

Convolution with 256 different filters in size of (3, 3)
kernel_regualarizer l2 with weight decay of 0.001
ELU activation function

Batch Normalization
MaxPooling by 2

Flattening the output from the last convolution layer

Fully Connected Layer with 128 units
ELU activation function

Fully Connected Layer with 256 units
ELU activation function

Fully Connected Layer with 100 units (number of classes/categories)

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 30, 30, 32)        896
_________________________________________________________________
batch_normalization (BatchNo (None, 30, 30, 32)        128
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 13, 13, 128)       36992
_________________________________________________________________
batch_normalization_1 (Batch (None, 13, 13, 128)       512
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 128)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 4, 128)         147584
_________________________________________________________________
batch_normalization_2 (Batch (None, 4, 4, 128)         512
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 2, 2, 256)         295168
_________________________________________________________________
batch_normalization_3 (Batch (None, 2, 2, 256)         1024
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 1, 1, 256)         0
_________________________________________________________________
flatten (Flatten)            (None, 256)               0
_________________________________________________________________
dense (Dense)                (None, 128)               32896
_________________________________________________________________
dense_1 (Dense)              (None, 256)               33024
_________________________________________________________________
dense_2 (Dense)              (None, 100)               25700
=================================================================
Total params: 574,436
Trainable params: 573,348
Non-trainable params: 1,088
```

Fig. 5. Model Summary.

C. *Choosing the best hyperparameters*

I have tried dozens of time to get the model with the maximum accuracy and this required tuning of hyperparameters as follows: Learning_rate: I have used the default leraning_rate of the ADAM optimizer. However, I have reduced the learning rate by a factor of 0.1, if my model stopped learning and shows a constant value for validation loss for the last 10 epochs using ReduceLROnPlateau. This helped me reach the minima.
Number of epochs = 100
Steps_per_epoch = 400

D. *Cost Function and Optimizer*

Cost function: Since, this a muti-class (100-class) classification problem, I have used a sparse_categorical_crossentropy loss function, and model is evaluated using the metric 'accuracy' against the validation set.

Optimizer: I have tried the SGD, ADAM and ADAGRAD optimizer for my model, with the best accuracy achieved through ADAM.

E. *Callbacks*

ReduceLROnPlateau: This callback was used to reduce the learning rate once the model stops learning

EarlyStopping: This was used to stop the training process, if a validation accuracy over a certain period (number of epochs) is constant

IV. RESULTS

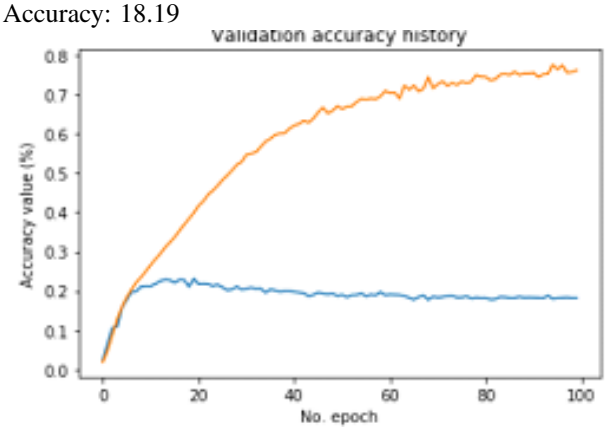
A. *Dataset*

Dataset: CIFAR-100 dataset. This dataset consists of colored natural images with 32*32 pixels. It consist of images drawn from 100 classes. The training and testing sets contain 50,000 and 10,000 images respectively, and we hold out 0.2% (8000) training images as a validation set. For preprocessing, we normalized the data and reshaped into the format required by the Tensorflow's tensor object to be fed to the convolution layer. We then use all the 50,000 training images and report the final test accuracy at the end of the training.

B. *Experiments and Performance Evaluation*

1. 3 Conv layers + 4 FC layers, activation function = ReLU

   - Accuracy: 18.19

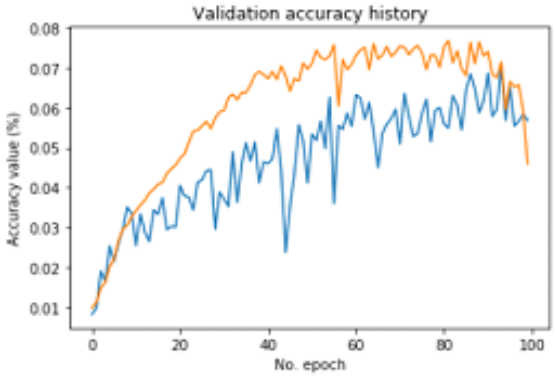


2. 4 Conv layers + 6 FC layers, activation function = ReLU

- Accuracy: 19.24

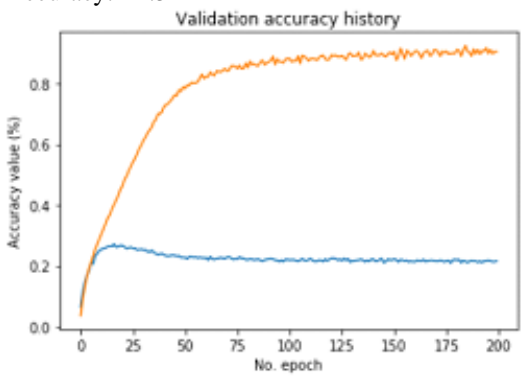
3. 4 Conv layers + 9 FC layers, activation function = ReLU

- Accuracy: 5.65



4. Normalized data, 4 Conv layers + 4 FC layers, activation function = ReLU,

- Accuracy: 22.32



5. Normalized data, 4 Conv layers + 4 FC layers, activation function = ELU

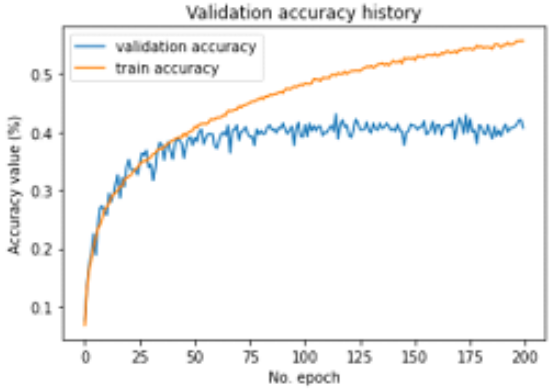
- Accuracy: 31.96

6. Normalized data, 4 Conv layers with Dropout(0.5, 0.2, 0.2)+ 4 FC layers, activation function = ELU

- Accuracy: 39.12

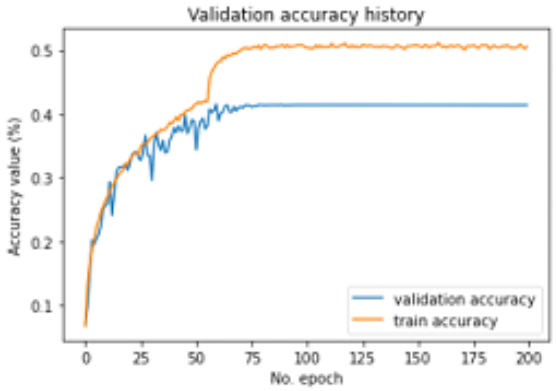
7. 3 Normalized data, 4 Conv layers with Dropout(0.5, 0.2, 0.2)+ 4 FC layers, activation function = ELU + Image Augmentation

- Accuracy: 40.12



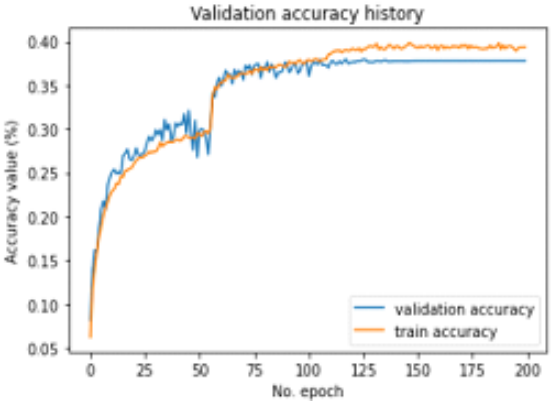
8. Normalized data, 4 Conv layers with Dropout(0.5, 0.2, 0.2)+ 4 FC layers, activation function = ELU + LROnPlateau

- Accuracy: 42.26



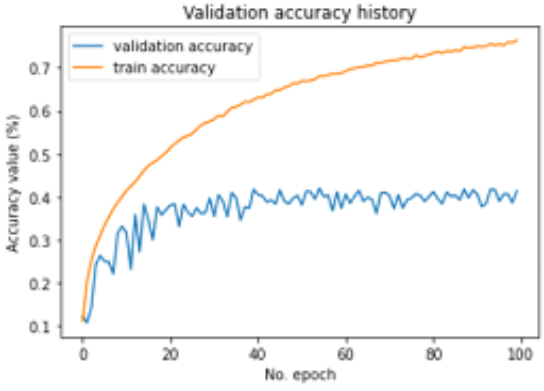
9. Normalized data, 4 Conv layers with Dropout(0.5, 0.2, 0.2)+ 4 FC layers, activation function = ELU + kernel regularizer

   - Accuracy: 34.72



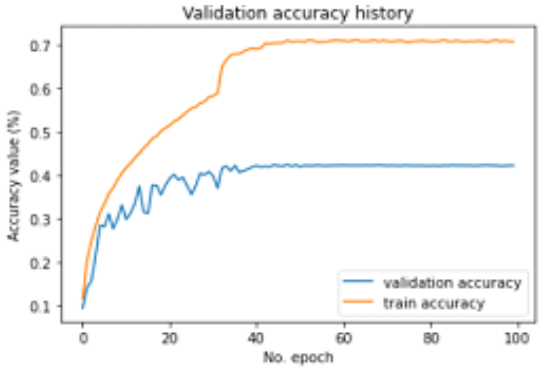
10. Normalized data, 4 Conv layers with Dropout(0.5, 0.2, 0.2)+ 4 FC layers, activation function = ELU + BatchNormalization

    - Accuracy: 42.12



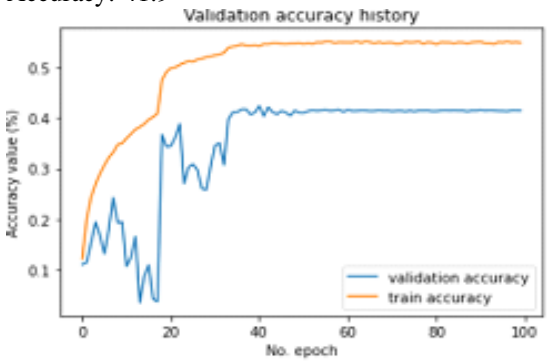
11. Normalized data, 4 Conv layers with Dropout(0.5, 0.2, 0.2)+ 4 FC layers, activation function = ELU + BatchNormalization + LROnPlateau

    - Accuracy: 42.37



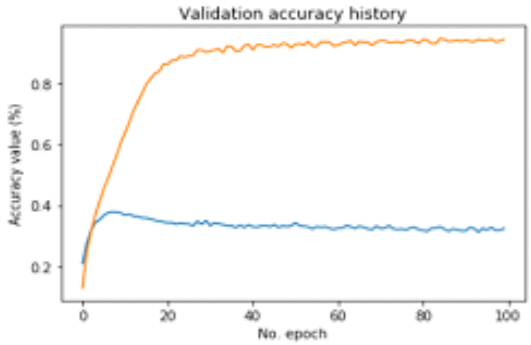
12. Normalized data, 4 Conv layers with Dropout(0.5, 0.2, 0.2)+ 4 FC layers, activation function = ELU + BatchNormalization + LROnPlateau + kernel regularizer

    - Accuracy: 41.9



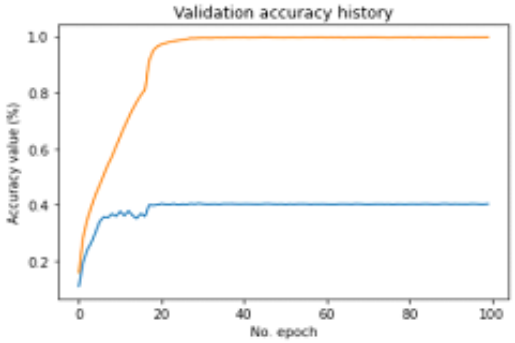
13. Normalized data, 4 Conv layers + 4 FC layers, activation function = ELU + Batch Noramalization

    - Accuracy: 35.13



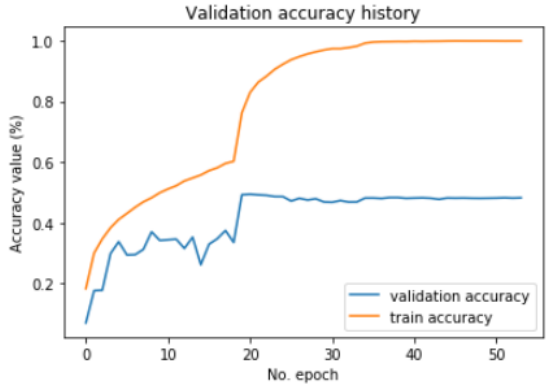
14. Normalized data, 4 Conv layers + 4 FC layers, activation function = ELU + Batch Noramalization + ReduceLROnPlateau

    - Accuracy: 40.29



15. Normalized data, 4 Conv layers + 3 FC layers, activation function = ELU + Batch Noramalization + ReduceLROnPlateau + kernel regularizer + EarlyStop

    - Accuracy: 48.39



C. *Results*

The experiments performed shows that increasing the depth of the network does not necessarily increase its accuracy. Instead the model may perform worse than the network with fewer layers (Experiment 2 and Experiment 5). Model performs better with Normalized data- we can see the accuracy increased by almost 3% if normalized data is fed to the convolution layers. The use of ELU activation function plays an important role here. The accuracy drastically increased by 10% just by changing the activation function from ReLU to ELU- this is because the ELUs have negative value which allows them to push the mean activation near to zero, and this speed up the learning process by bringing the normal gradient closer to the unit natural gradient because of a reduced shift effect. Accuracy jumped to 31.96% with the use of ELU activation function.
Looking at the graphs in the above table, the model was overfitting, and hence techniques like batch normalization, dropout and kernel regularizer were tried and tested with different combinations to avoid overfitting. It was clear that dropout does not have much effect after batch normalization- so when using Batch normalization, we can skip dropout. ReduceLROnPlateau was used to reduce the learning rate of the optimizer when the model stops learning. This helps in getting better accuracy.

Finally, a model with 4 Convolution layers and 4 Fully connected layers with activation function as 'ELU' was used to achieve an accuracy of around 48.39% with the help of batch normalization, kernel regularizer and ReduceLROnPlateau. Early stopping was implemented to stop the training process once the model stops learning at all and it starts to overfit, to improve model performance.

The below graphs demonstrate the training loss and the validation loss over the number of epochs with the final model:

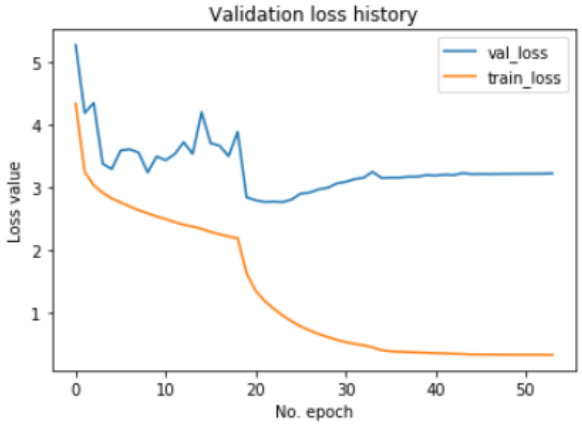


Fig. 6. Loss graph.

The below graphs demonstrate the training accuracy and the validation accuracy observed over the number of epochs with the final model:

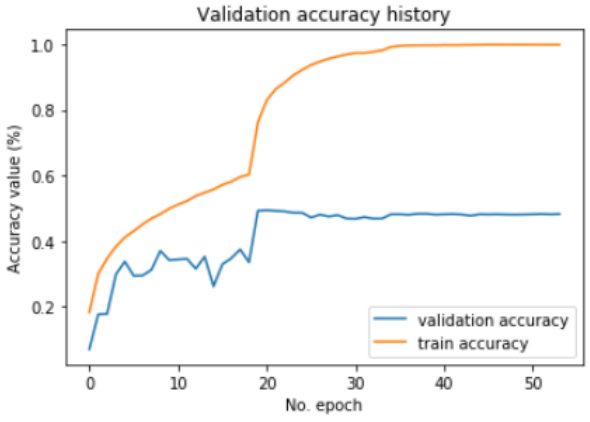


Fig. 7. Accuracy graph.

Fig. 8. Predictions by the model.

D. *Discussion*

1. Model performs better with Normalized data
2. Activation function plays an important role in model optimization. Accuracy drastically changed from 22.32% to 31.96% by just changing the activation function from ReLU to ELU.
3. Dropout doesn't play a major role in improving the performance of the model, if Batch Normalization is used-however vice-versa is not true
4. It is important to reduce the learning rate (usually by a factor of 0.1) once the model stops learning
5. Just adding layers does not necessarily improve the accuracy of the model, in stead it may degrade because of the exploding/vanishing gradient problem

V. CONCLUSION

In this project, various strategies were discussed to optimize the model. Although there are many complex architectures with larger and deeper networks like AlexNet, ResNet, etc. which were able to achieve a good accuracy, this model with just 4 convolution networks and 4 fully connected layers achieved state-of-the-art results. Since this project focuses on the strategies that can be used in any other network that can help them to optimize their model further, we believe that further gains in accuracy may be obtained by more detailed tuning of hyperparameters and learning rate schedules.

VI. ACKNOWLEDGEMENTS

I would like to express my deep gratitude to Professor Dr. Jerome J. Braun, my supervisor, for his patient guidance, enthusiastic encouragement and useful critiques of my work. I would also like to thank him for his advice and assistance in keeping my progress on schedule.

VII. REFERENCES

1. https://www.tensorflow.org/tutorials/images/cnn
2. https://www.tensorflow.org/tutorials/images/cnn
3. https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in
4. https://towardsdatascience.com/build-your-own-convolution-neural-network-in-5
5. https://cs231n.github.io/convolutional-networks/
6. https://ml-cheatsheet.readthedocs.io/en/latest/activation_\functions.html#elu
7. https://towardsdatascience.com/implementing-batch-normalization-in-python-a04
8. https://arxiv.org/pdf/1902.02771.pdf
9. https://arxiv.org/pdf/1511.07289.pdf
10. https://arxiv.org/pdf/1608.06993v5.pdf