Implementing Convolutional Neural Networks with Keras

Hunor Laczko and Smriti Joshi

I. INTRODUCTION

Deep learning has proved to be a useful tool capable of modelling complex functions present in real-life problems and providing accurate results. This laboratory report presents the analysis of the performance of supervised convolutional neural networks. These networks are evaluated on two major datasets for multiclass classification. This report has the following structure: Tools used to train and evaluate the network are discussed in Section II, analysis of the data is presented in Section III, the methods and implementations are discussed in Sections IV and the results are summarized in section V. Further, the report is concluded and potential future work is discussed in section VI.

II. PRELIMINARY

This section discusses the cost function and metrics used for training and evaluating the network.

A. Cost function: Cross Entropy

Cross entropy is a popular choice for loss function, it measures the similarity between two vectors or distributions. In a typical multiclass classification application it is used to measure the similarity between the output of a softmax layer (the probability of object belonging to each class) and the one-hot-encoded ground truth label. Mathematically, it is represented by Equation 1.

$$CE = -\sum_{i=1}^{C}(A_i \log B_i) \tag{1}$$

where C is the total number of classes, *A* is the true label, *B* is the output probability.

B. Evaluation metric

1) Confusion matrix: Confusion matrix is a table used to present the performance of binary classifiers in the case where true labels are known. False negative, also called Type II error, refers to the case when the training example belongs to positive class but is incorrectly classified as negative class. False positive, called as Type I error, refers to the case when a training example belongs to negative class but is incorrectly labelled as positive class.

2) Accuracy: Accuracy is the number of instances classified correctly over the total number of instances. Mathematically, it is given by:

$$Acc = \frac{TP + TN}{TP + FP + TN + FN} \tag{2}$$

This is not useful for imbalanced datasets because the majority class is usually predicted correctly because of higher number of training examples and leads to higher accuracy.

C. Optimizers

1) Stochastic Gradient Descent (SGD): This algorithm is an iterative method to optimise the loss function. It introduces stochastic approach in gradient descent by replacing the gradient with the estimation of gradient from randomly selected batch of data. This decreases the computational burden achieving faster iteration.

2) Adam: Adam optimiser is also an iterative method and extends on the idea of RMSProp. The learning rate in this case is adapted not only on the basis of mean of gradients but also on the average of uncentered variance. In the paper proposing adam, the optimizer is presented as increasing the speed of training considerably.

D. Activation functions

Activation functions are popular to introduce non linearity in the network to fit complex distributions. In the networks, they are used to determine if the neuron should be fired and to what extent. Two kinds of activation functions are used in this assignment. They are explained in more detail below.

1) ReLU: ReLU stands for Rectified Linear Unit. It has advantage over sigmoid/tanh function mainly because they tend to saturate at the extreme values and only are sensitive to changes in the middle of their respective ranges. ReLU solves this problem by using a different kind of non-linear function. Mathematically, it is defined as:

$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

2) Softmax: Softmax function can be thought of as extension of logistic function to multiple dimensions. In this assignment, it is used as the last layer of multiclass-classification network to predict the probability distribution over all classes. Mathematically, this can be obtained with the following equation:

$$S(x) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{4}$$

where the summation is over probabilities for all inputs.

III. DATASET

The datasets used in this assignments are MNIST and CIFAR10. For both the datasets, the training set is split into training and validation set with a factor of 0.2.

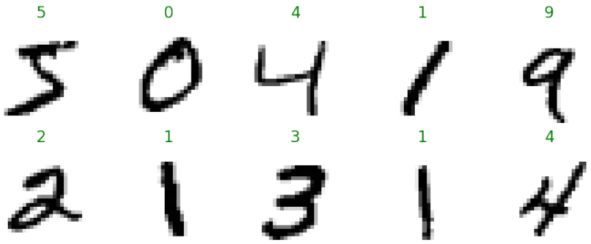


Fig. 1: Subset of MNIST dataset used for training

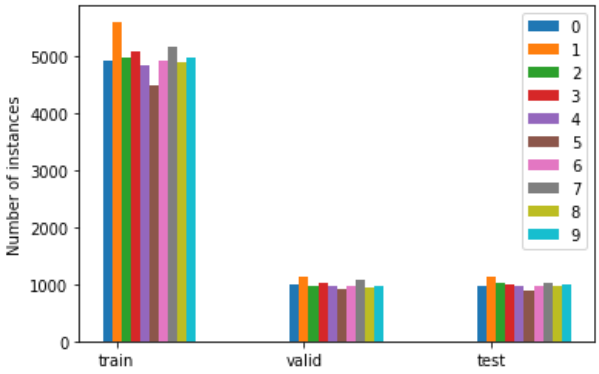


Fig. 2: Graph showing distribution of data in MNIST Dataset

A. MNIST

The dataset used for training and evaluating the network architecture described in section IV-A.1 MNIST[1] stands for Modified National Institute of Standards and Technology. This dataset contain 60,000 images of size 28 x 28 square pixels containing handwritten numbers from 0 to 9. The classes for these numbers are also 0 to 9 respectively. This can be visualised in Figure 1. The title of each image displays the ground truth label. For each class, there are roughly equal number of instances in train, validation and test set as seen in the bar graph presented in Figure 2.

B. CIFAR10

The CIFAR-10 dataset[2] consists of 60000 32x32 colour images in 10 classes, with 6000 images per class (uniform distribution). There are 50000 training images and 10000 test images. A few example images can be seen in Figure 3 for each class. It is important to mention that there is no overlap between automobile and truck classes. Automobiles is used to define vehicles like SUVs, sedans etc while truck refers to images of big trucks. The dataset is more challenging than MNIST because of color images and varying backgrounds

Fig. 3: Subset of CIFAR10 dataset used for training

IV. METHOD AND IMPLEMENTATION

This section discusses the methods for networks implemented in this assignment.

A. Architecture

Two types of architecture are discussed below for respective datasets. It is worth mentioning that the activation functions used for hidden layers and for output layer are 'ReLU' and 'softmax' respectively in all cases.

1) MNIST classification: For Mnist classification, the architecture used is displayed in Figure 4. In the beginning, it has one convolution layer followed by maxpooling of size 2. This block is followed by two convolutional layers and one more maxpooling layer. Finally there are two dense layers which yield the output. The input to this architecture is image and its one hot encoded label.

2) CIFAR10 classification: Two architectures are tested for this dataset. The first one is the same architecture that was previously used for MNIST dataset. However, it was modified to prevent overfitting by adding a dropout layer for the first fully connected layer. The second architecture adopted to achieve a higher accuracy contains VGG blocks. VGG blocks are very common and simple methods for image classification. They contain two convolutional layers with same padding and a max pooling layer. To further improve the results, batch normalization is also tried with this architecture. After a little experimentation the network architecture shown in Figure 5 is finalised to train on the current dataset. Here, each convolution block refers to two convolution layers, each followed by Batch Normalisation, followed by max pooling of size 2 and dropout of factor 0.2. Similarly, Dense Block refers to a dense layer followed by batch normalisation and dropout with factor 0.2.

B. Data Augmentation

ImageDataGenerator from keras preprocessing library is used to generate the augmentation of images while training the network. After experimentation with available data augmentations, three kinds of augmentation were chosen for the current task: width_shift_range, height_shift_range and horizontal_flip. These transformations are randomly generated. Such a random generation of 8 images for 80th training image is shown in Figure 6. The first images with a black border represents the original image. In the subsequent images, it can be observed that 7 out of 8 images are flipped horizontally. Further, the cat seems closer/father from the camera due to horizontal shift. Similarly, sometimes more/less ground is visible due to vertical shift.

C. Transfer Learning

Transfer learning refers to the process of using pre-trained weights to initialise the network. In theory, it means applying knowledge gained in one problem to solve different but related problem. This report uses different

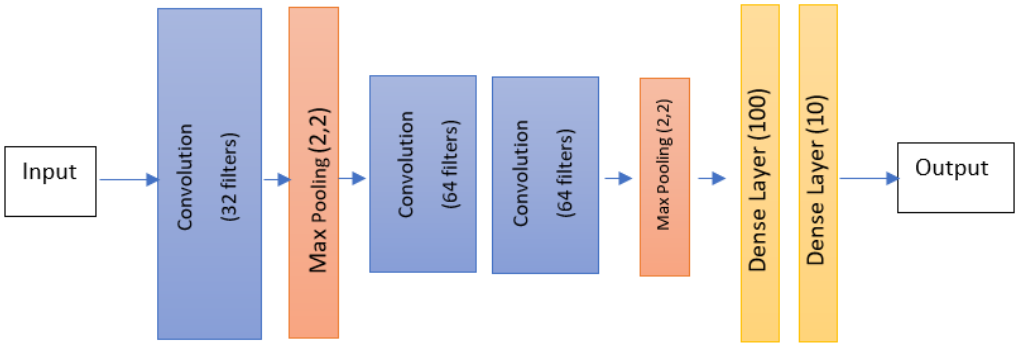


Fig. 4: CNN architecture for classification of MNIST Dataset

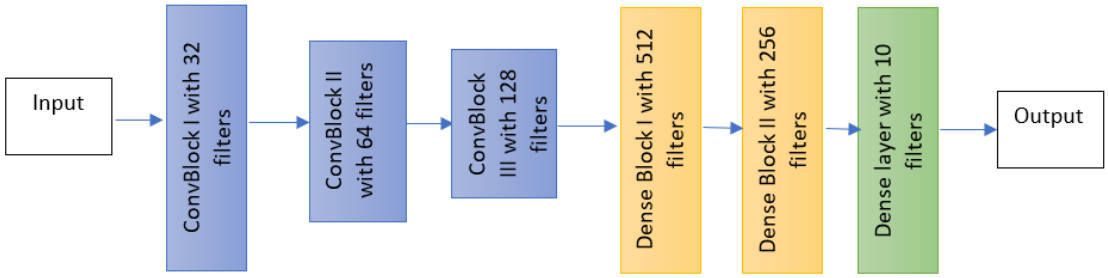


Fig. 5: CNN architecture for classification of CIFAR10 dataset

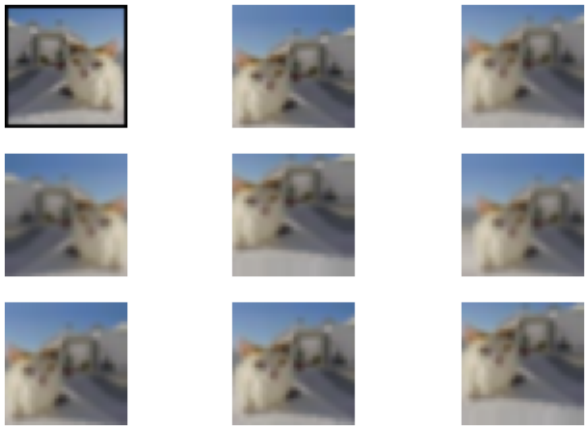


Fig. 6: Randomly generated samples from ImageGenerator with different deformations

architectures trained on Imagenet. Taking the base model as known architecture, the final classification layers are replaced by new fully connected layers and fine tuned for the CIFAR10 dataset. This pre-trained models are obtained from keras.applications module. Two kinds of architectures were used for this purpose:

1) VGG19: VGG19[1] is a variant of VGG model which in short consists of 19 layers (16 convolution layers, 3 Fully connected layer, 5 MaxPool layers and 1 SoftMax layer). This network was proposed during ILSVRC challenge and is popularly used for image classification. There are other variants of the same namely VGG11, VGG16 etc.

2) ResNet50: ResNet was also introduced as a part of ImageNet challenge. ResNet50 is a variant of ResNet model which has 48 Convolution layers along with 1 MaxPool and 1 Average Pool layer. It has 3.8 x 10^9 Floating points operations. This network is appreciated in the community because it enabled the use of deeper networks by dealing with vanishing gradient problem through the concept of skip connections.

D. Training

Bases on experiments the following hyperparameters were chosen during this lab. For batch size 64 was used. For

[1]https://iq.opengenus.org/vgg19-architecture/

learning rate, a value of 0.01 was used with a momentum of 0.9 with SGD optimiser. This helped achieve convergence faster. Even though it oscillated a bit while training, there was no significant effect on the results. For transfer learning, Adam optimizer was also used along with SGD with a lower learning rate of 0.0001 since only fine tuning was needed. Since the models were prone to overfitting on the CIFAR dataset, an early stopping mechanism was used with a patience of 20 or 50.

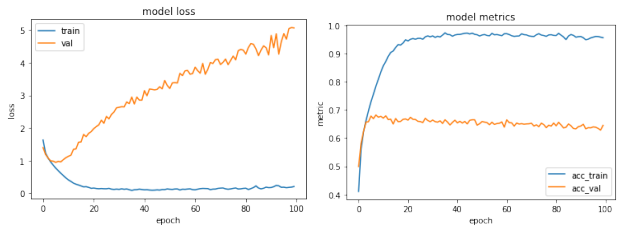
V. RESULTS

A. MNIST Classification

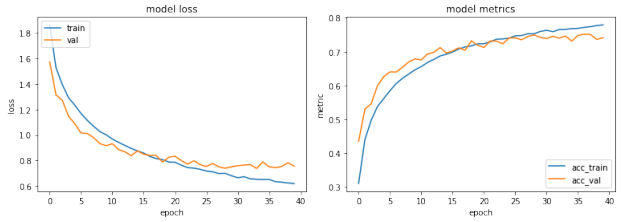
The use of a CNN was already explored in the previous lab. There with CNN a significant performance gain was achieved with 99.37% over 95.85% which was achieved with a single hidden layer network. More detailed results can be found in the previous lab report.

B. CIFAR10 Classification

1) Simple CNN: First, the same CNN was tried on the CIFAR dataset as the one used for the MNIST dataset. Running with the same parameters this network only achieved an accuracy of 63.1%. Inspecting the training loss graph as shown in Figure 7a, it was noticed that the model started to overfit very soon. To counteract this, two dropout layers were added before and after the fully connected layer. with this the accuracy increased significantly to an value of 74.24%. This regularisation effect due to drop out is visible in Figure 7c where validation loss follows the training loss closely. These results can be seen in Table I.



(a) Loss (b) Metric



(c) Loss with dropout (d) Metric with dropout

Fig. 7: Plots showing loss and metric over increasing number of epochs for simple CNN architecture for CIFAR10

2) Using VGG blocks: Using VGG blocks proved effective for this task. With only three VGG blocks and two fully connected layers an accuracy of 81.89% was achieved. This model contained dropout layers after each convolution block and fully connected layer which helped avoid overfitting. Because this model was less prone to overfitting the early stopping patience was increased to 50. Adding batch normalization after each convolution and fully connected layer further improved the model, resulting in an accuracy of 84.42%. This is due to the reguralization effect of combining both dropout and normalization layers. These results can be seen in Table I. With this method an improvement of 10% was achieved.

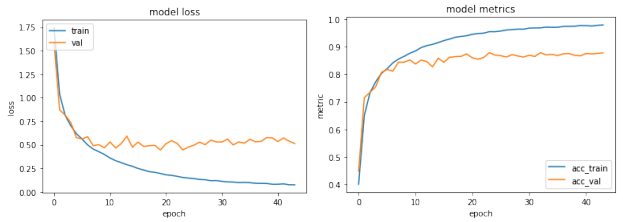
Fig. 8: Confusion matrix for VGG architecture with added batch normalisation

Fig. 9: Misclassified 'cat' class by VGG architecture with added batch normalisation. The value in the bracket indicates the confidence in the classification.

Figure 8 shows the confusion matrix obtained for this network architecture. It can be observed that 'cat' is classified the worst closely followed by 'bird' and 'dog'. A huge number of instances for 'cat' is confused with 'dog' and vice versa. This is understandable as both are four legged animals with similar variations in colour. A few instances of missclassified 'cat' class can be seen in Figure 9. For most of the images, the confidence for classification is high indicating that the network performs pretty bad for certain images. For instance for image 12, the network is 98% confident that the image belongs to 'deer' class. This is probably because images of deers are usually in green environments and they have brown colour with white spots which is similar to the given image. Further looking into confusion matrix, it

Network	Accuracy (%)
Simple CNN	63.10
Simple CNN + Dropout	74.24
VGG blocks + Dropout	81.89
VGG blocks + Dropout + Batch norm	**84.42**

TABLE I: Results for simple CNN and VGG blocks based CNN



(a) Loss (b) Metric

Fig. 10: Training loss and metric for best performing network

Fig. 11: Confusion matrix for Transfer-VGG19 + batch_norm + augmentation

is also interesting to see that 'automobile' and 'truck' are confused with each other. This is also expected due to similar structures (four-wheeled) and similar backgrounds (roads, highways) for the two classes.

3) Transfer Learning: In this experiment a pretrained model was used taking its feature extraction part and adding new fully connected layers to it.

A Resnet50 and VGG19 models were used here pretrained on the ImageNet dataset. Since only fine-tuning was performed the learning rate was lowered to 0.0001 and the early stopping was used with a patience of 20 to avoid overfitting. Two optimizers were used, namely SGD and Adam, but Adam performed better early on so the majority of the experiments were done with it. The results can be seen in Table II. It can be seen that when training with SGD the model performs on the level of the simple CNN with only 1% improvement but training with Adam results in 80.85% accuracy which is a significant gain but it is still under the VGG block based methods performance.

Seeing how the VGG block based model performed well previously, next the VGG19 model was used for transfer learning. Based on the previous experiments using Batch normalization was beneficial for the performance they were used here too. With VGG19 using Adam optimizer and the batch normalizations an accuracy of 84.97% was achieved slightly outperforming the VGG block based method with 0.5%.

Network	Accuracy (%)
Resnet50 + SGD	75.52
Resnet50 + Adam	80.85
VGG19 + Adam + Batch norm	**84.97**

TABLE II: Results for transfer learning

4) Data augmentation: Two sets of data augmentations were tested. First, every image was flipped both vertically and horizontally and rotated in the range of 20 degrees. Early experiments showed that this combination was not beneficial, it even degraded the previous methods' performance. On the VGG block based method with dropout a decrease of 5.2% was seen. This was slightly increased when using only the flipping augmentation without rotation, but it was still underperforming the previous method by 3.7%. Using transfer learning with these augmentation had similar results, Resnet trained with both SGD and Adam decreased by 1.5%.

After this, another augmentation was tried, which included only the horizontal flip but added shifting of the image both horizontally and vertically. With this significantly better results were achieved as it can be seen in Table III. The table shows the improvements over experiments without data augmentation. The columns with one value represent the version with the data augmentation which were not tested otherwise since previous experiments proved that batch normalization did not have any significant effect when used with transfer learning only.

All models accuracies increased when using these augmentations by several percents. The most notable ones are the VGG block based method using dropout and batch normalization and the VGG19 transfer learning model. The first one achieved almost 3% increasing reaching 87.11%. This was only outperformed by the later one, with a performance of 88.11%. As the table shows, the same method without batch normalization achieved only 0.1% lower performance which could be due to the random initialization. Figure 11 shows the confusion matrix for best performing network. Compared to the previous confusion matrix in Figure 8, better performance is observed. Except 'dogs', 'cats' and 'birds', the classification values is over 90% for all the classes. The network still gets confused between 'cats' and 'dogs' classes. However, compared to 68% in the previous case, 76% cats are classified correctly in this scheme which is a considerable increase. In Figure 12. the top 25 worst classified images are shown. Worst classified is interpreted as the wrong predictions with the highest confidence. In these

Network	Accuracy (%)
VGG blocks + Dropout	81.89 ->83.51
VGG blocks + Dropout + Batch norm	84.42 ->87.11
Resnet50 + SGD	75.52 ->82.27
Resnet50 + Adam	80.85 ->84.74
Resnet50 + Adam + Batch norm	84.5
VGG19 + Adam	88.01
VGG19 + Adam + Batch norm	84.97 ->**88.11**

TABLE III: Results with data augmentation

cases the confidences were all over 99%. As seen in the figure all cases represent hard scenarios. It is interesting to note that there are wrongly labeled instances in the dataset as seen in the top-left image of the figure, which is labeled as a cat but it is green and also gets classified as a frog which is probably the right label.

The training process for the best performing network can be seen in Figure 10. It shows that the training stopped at epoch 44 due to the early stopping criteria. This was beneficial, since the validation loss and accuracies were not improving and the training loss and accuracy already converged at this point too, so training longer would have caused overfitting.

These experiments prove the importance of having databases of large size for training. It is also worth noting that while data augmentation improves the models, their training times also significantly increase, for instance in the VGG block based method the training time increased from 6s/epoch to 27s/epoch (running on RTX 2070). Moreover, while achieving almost the performance the transfer learning model had 30 times more parameters than the custom VGG block based method. Although, interestingly this did not show in the training times. This was probably because the bottleneck of the training was not the computation capacity but rather the memory bandwidth.

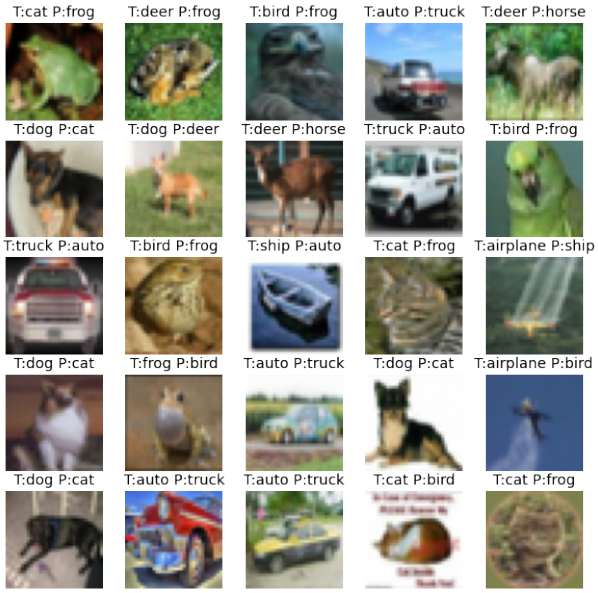


Fig. 12: Worst classified images

VI. CONCLUSIONS

In conclusion, this report investigated the performance of CNNs on two datasets and compared it with training with simple fully convolutional layers. Further, techniques like transfer learning and data augmentation were used to further improve the values of metrics. For MNIST dataset, a simple CNN achieves 99.37% accuracy. For CIFAR 10, a decent accuracy of 88.1% is achieved considering that relatively simple solutions were used in terms of architectures, loss functions and optimizers. The current state of the art accuracy is 99.5% [2] obtained via sophisticated approaches which can be investigated in the future.

REFERENCES

[1] LeCun, Y. & Cortes, C. (2010). MNIST handwritten digit database
[2] Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, 2009.

[2] https://paperswithcode.com/sota/image-classification-on-cifar-10