PROJECT REPORT

Asraa Amin

JULY 20, 2019

UNIVERSITY OF GLASGOW

Introduction:

A Convolutional Neural Network **(**CNN**,** or ConvNet**)** is a special architecture of multi-layer artificial neural networks, that is commonly used for image classification. In this project, CNN is used as an image classifier for the **CIFAR-10 dataset. CIFAR-10 dataset** comprises of 60,000 low-resolution (32x32) colour images. It consists of 10 different classes with 6000 images in each class, which are airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. These 10 classes can be classified to two classes, animals and modes of transport. Animal class includes bird, cat, deer, dog, frog, and horse. While transport includes airplane, automobile, ship and truck. Our goal is to develop a CNN that differentiates between images of animals and forms of transport.

Data Manipulation:

The data divided into training data with 50000 images and test data with 10000 images. The training data is also divided into training dataset with 40000 images and validation dataset with 10000 images. For the training, test and validation datasets, the 10 classes are combined into two classes as mentioned before. For simplicity and faster convergence, the training, test and validation are normalized (divided by 255) so the pixel range is bounded by 0 and 1.

Data Model:

Part1:

A Simple CNN is built from stacking several types of layers:

- First layer: A convolution layer with 3x3 kernel size and 16 filters.
- Second layer: Flatten layer to flatten the input.
- Third layer: Fully connected layer with 512 units.
- Final layer: Dense Layer with a softmax activation function.

This CNN is trained on the training dataset with 40000 images using TensorFlow session. The input image to this CNN is a tensor with 32 pixels width, 32 pixels height and 3 channels representing RGB (red, green, blue) colour intensities. While training, images from both the classes (animals / transport) are fed to the first convolutional layer. After the convolutional layer, the output is flattened, and two fully connected layers are added at the end. In the first fully connected layer, two activation functions performance are examined, the rectified linear unit (ReLU) and Hyperbolic Tangent activation function (Tanh) function. The second fully connected layer has only two outputs which represent the probability of an image being an animal or a transport.

The main objective of the training process is to get the best possible values for the network parameters weights and biases such that the output layer gives out probability of transport as 1 for all images of transport and probability of animals as 1 for all images of animals. To find the best set of parameters and increase our network accuracy, optimizers are used to minimize the cost function. Cross-entropy is used to measure the cost and the performance of two optimizers which are Adaptive Moment Estimation (Adam) optimizer and Root Mean Square Propagation optimizer (RMSprop). Adam optimizer is the most popular optimizer as it builds upon RMSProp and adds momentum. The training data is divided to 40 batches and trained for 250 epochs.

To find an appropriate learning rate, the network is trained with different learning rate values starting with a small value of 1e-10 and increase this value exponentially each step till it reaches 1 [1]. The training loss is recorded at each step. Also, Tensorboard is used to visualize the progression of network training over time (the learning rate and the training loss at each step). The training loss and learning rate plots are analysed to find suitable learning rate for the network. (see Figure 5). To find suitable number of epochs for our model, the learning rate from the previous step is used to train the model and

plot the training/validation loss for every step. From the plot, the proper number of epochs is equivalent to the minimum validation loss.

Part2:

In this part, Keras training utilities is used to train the data. Our basic network is:

1- Convolution layer with 16 feature maps using a filter 5x5 and stride 1 (**16C5**)
2- Flatten layer to flatten the input.
3- Fully connected dense layer with 512 units (**512**)

Adam optimizer is used with the binary cross entropy optimization score. A training rate scheduler is used which take the epoch index and current learning rate and return a new learning rate. The data trained in batches with 128 batch size. To enhance the CNN performance and increase the classifier accuracy, many changes can be done to the network such as:

- Adding Convolutional layers to increase the number of feature maps for each subsequent pair (specify optimal filter size, strides and zero padding)
- Adding regularisation layer to prevent our network from overfitting thus helping our network generalize better (dropout and batch normalization).
- Add pooling layer (**P**) after every convolutional layer to control overfitting by reducing the number of parameters and computation in the network.
- Instead of adding pooling layer, CONV layer with larger stride is used to reduce the size of the representation. This method is better than pooling as it is important in training good generative models.
- Increase the training dataset by using data augmentation. Data augmentation is a method used to increase the training data size by generating modified versions of images in the dataset.

We started from the basic model and used the method in [2] to find the best model. To find the best number of Conv layers, number of feature maps for each layer, number of units for dense layer and amount of dropout, the steps below are followed:

1- First, we examined the number of pairs of convolution-subsampling (Convolutional layer +pooling layer) whether one two or three have the best training/validation accuracy. We can't use fourth layer as the output of the third layer will be 1x1. The convolutional layers with stride of 1 is used to transform the input volume depth-wise while leaving the down-sampling to the pooling layers.
   - (**[16C5-P2]-256-2**)
   - (**[16C5-P2]-[32C5-P2]-256-2**)
   - (**[16C5-P2]-[32C5-P2]-[64C5-P2]-256-2**)

2- Then, by using different values of feature maps for the convolution-subsampling layers, the best number of feature maps for each layer is determined in terms of training/validation accuracy (**8, 16, 24, 32, 48 or 64**).
3- Different number of dense layers and dense units are examined to find the best one (**16, 32, 64, 128, 256, 512, 1024 or 2048**).
4- After each layer, the amount of dropout should be used to enhance the CNN performance and prevent overfitting (**0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6 or 0.7**)
5- Lastly, different changes to the model are examined to improve the accuracy:
   - Conv layer of size 5x5 is exchanged with two Conv layers of size 3x3 (like in VGGNet [3]).

- Examine if Batch normalization will enhance the network performance.
- The pooling layers are replaced with Conv layer with larger stride to subsample so the network will be all Convolutional network [4].
- Examine if Adding data augmentation will improve the training/validation accuracy.

Results:

Part1:

Performance evaluation: Figure 1, Figure 2, Figure 3 and Figure 4 show the progress in the validation accuracy and loss function over time using Tensorboard for different optimizers and activation functions. Using the test data, the model accuracy is measured. Table 1 shows the performance of each optimizer for each activation function. For the same learning rate, number of epochs and batch size, it is noticed that Adam optimizer outperform RMSProp slightly. Also, it is noticed that ReLu activation function perform better than Tanh.

Finding appropriate learning rate and epochs: From Figure 5, the loss plot with the training steps, the loss starts at high value and remains almost constant for the first 3000 steps, then it starts to decrease. The training step where loss falling fast (steepest point) is the training step with the appropriate learning rate. From the learning rate plot, this training step is used to find the corresponding learning rate. By using this learning rate and train the network for long time, the most appropriate epochs number is identified. The resultant learning rate and number of epochs are shown in Table 2. Again, Adam optimizer outperform RMSProp. (Figure 6 and Figure 7).

Table 1: Performance Evaluation of different optimizers/Activation Functions

Optimizer	Activation Function	Number of Epochs	No of Batches	Learning rate	Accuracy
Adam	ReLU	250	128	0.0001	0.8773
Adam	Tanh	250	128	0.0001	0.8715
RMSProp	ReLU	250	128	0.0001	0.8362
RMSProp	Tanh	250	128	0.0001	0.8397

Table 2: Performance of the models with appropriate learning rate/No of epochs

Optimizer	Activation Function	Number of Epochs	Batch size	Learning rate	Accuracy
Adam	ReLU	600	128	4.3485e-4	0.9022
RMSProp	ReLU	600	128	7.3863e-4	0.8963

Part 2:

Following the steps above, the resulting network from each step:

1- The best number of pairs of convolution-subsampling (Convolutional layer +pooling layer) is two so the resulting network (**[16C5-P2]-[32C5-P2]-256-2**)
2- The best feature map for each layer is 32,64. (**[32C5-P2]-[64C5-P2]-256-2**)
3- The dense units are 64 (**[32C5-P2]-[64C5-P2]-64-2**)
4- After each convolutional-subsampling layer, 40% and 20% dropout are the best. We used 40%.
5- By trying the changes above, the results are:

- The Conv layer of size 5x5 is exchanged with two Conv layers of size 3x3 (**[32C3-32C3-P2]-[64C3-64C3-P2]-64-2**
- Batch normalization doesn't enhance the validation accuracy.
- Exchanging the pool layer with Conv layer with larger stride doesn't enhance network performance.
- Increasing the training dataset doesn't improve the validation accuracy.

The last model is (**[32C3-32C3-P2]-[64C3-64C3-P2]-64-2** with test accuracy **%96.03.**

Discussion and Conclusion:

- Adam optimizer performs better than RMSProp. Adam and RMSProp are both adaptive learning rate method. They are very similar except that Adam adds bias correction and momentum to RMSProp. As gradients become sparser near the end of the optimization, this bias correction makes Adam to outperform RMSProp slightly.
- Tanh activation function is a zero centred function that suffer from vanishing gradient problem which result in being too slow to reach an accurate prediction. Thus, it degrades the accuracy and performance of the model as noticed from our result. The ReLu activation function is non-linear function that allows the network to converge very quickly. It has 6 times improvement in convergence from Tanh function. Hence it rectifies vanishing gradient problem and result in better accuracy (as seen in the results). Its drawbacks that it should only be applied to the hidden layers and in certain cases it may result in dead neurons.
- We noticed that using Small batch size results in fast learning but unsteady learning process which leads to high variance in the accuracy. While using large batch size results in slow learning process but leads to stable model, lower variance in the accuracy. When we increased the batch size to 1024 and higher, we got higher accuracy. But larger batch size usually results in high generalisation error [5].
- From table 1 and table 2, as the number of epochs is increased from 250 to 600, the network performance goes from underfitting to optimal. If the number of epochs is increased too much, the model will start to memorise the training dataset instead of learning it (overfitting).
- Using multiple Conv layers with smaller size kernels instead of one Conv layer with larger size, increases the network depth. Thus, more complex features are learned, and the accuracy increased.
- Batch normalization is not recommended with dropout as it reduces the generalisation error due to the noise in the statistics used to normalize each variable. Also, it is recommended for deep network with more than two layers. Because we used high dropout and our network is not very deep, the network performance is not improved by adding batch normalisation.
- Every time the networked is run, we get slightly different result. To get accurate accuracy, we must run the network for hundreds of times and take the average

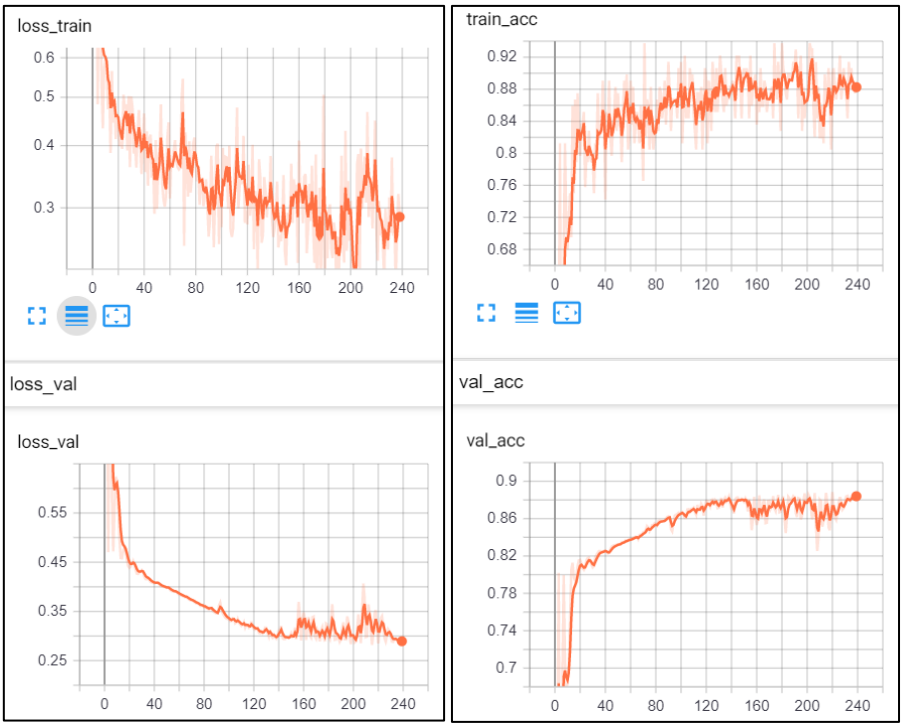

Figure 1: Adam optimizer loss function and Accuracy (Training and Validation) with ReLu

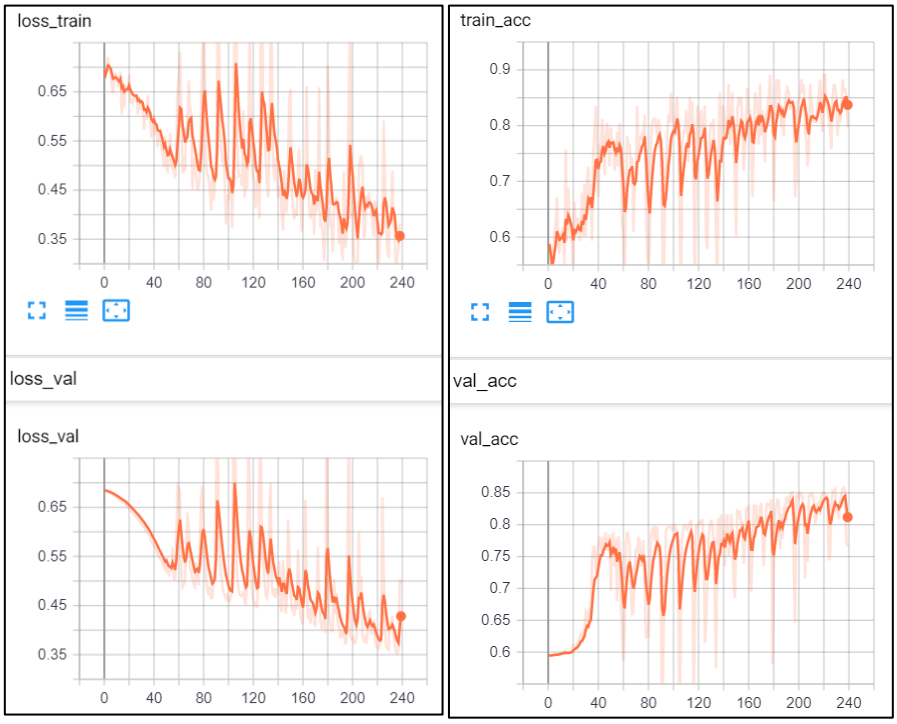

Figure 2: RMSProp optimizer loss function and Accuracy (Training and Validation) with ReLu

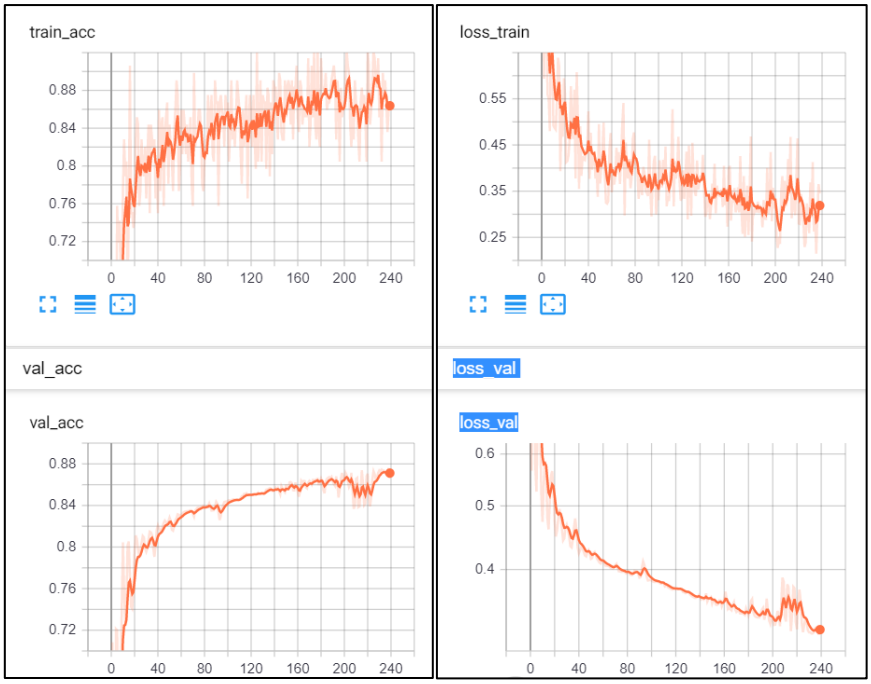

Figure 3: Adam optimizer loss function and Accuracy (Training and Validation) with Tanh

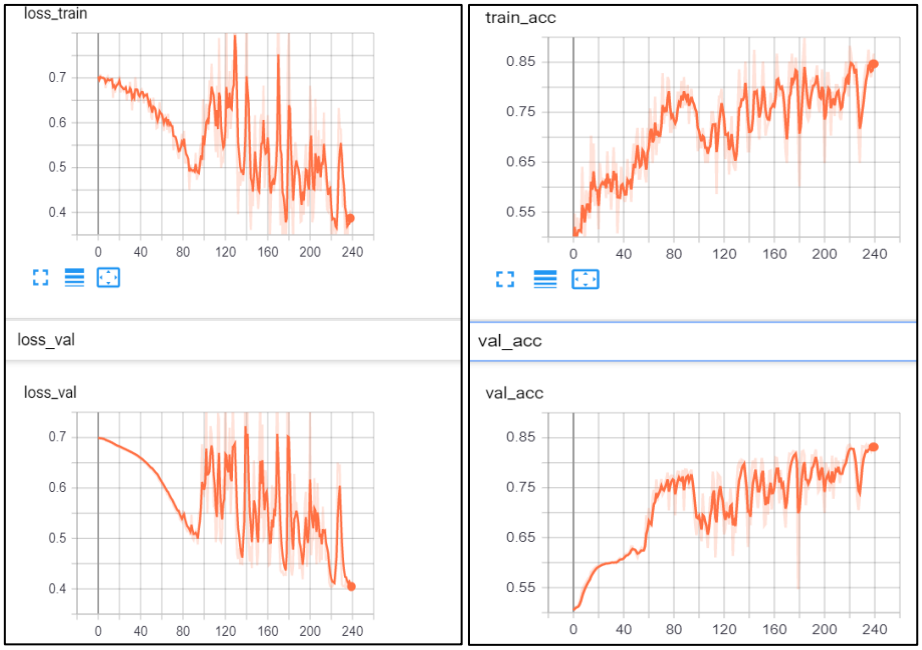

Figure 4: RMSProp optimizer loss function and Accuracy (Training and Validation) with Tanh

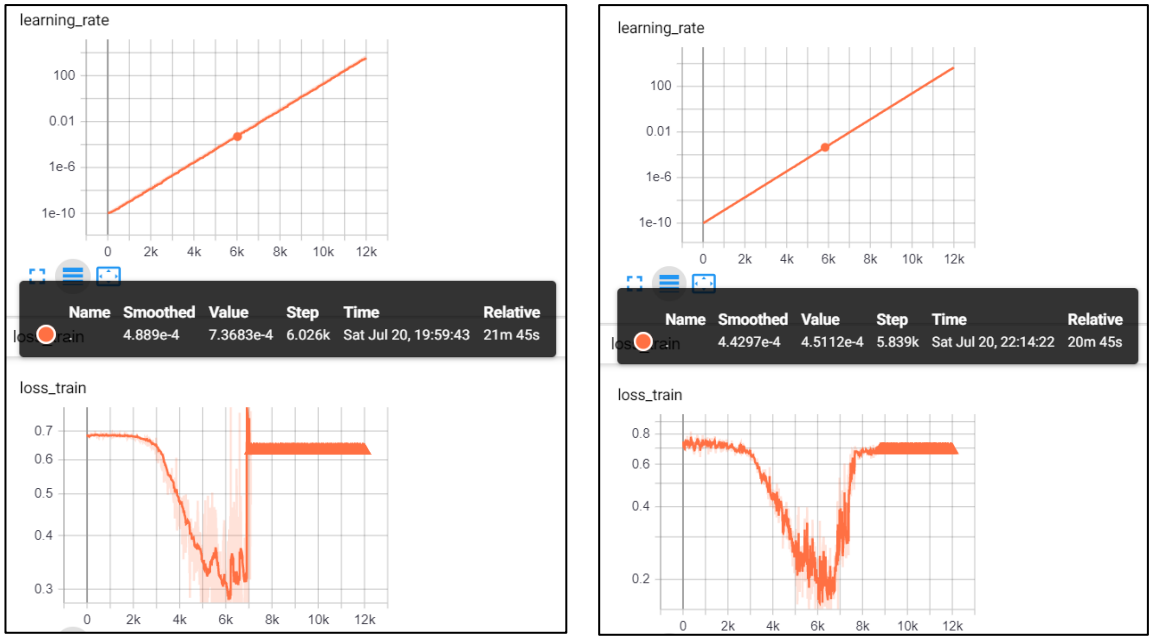

Figure 5: Finding appropriate learning rate for RMSProp (left) and Adam (right) optimizers

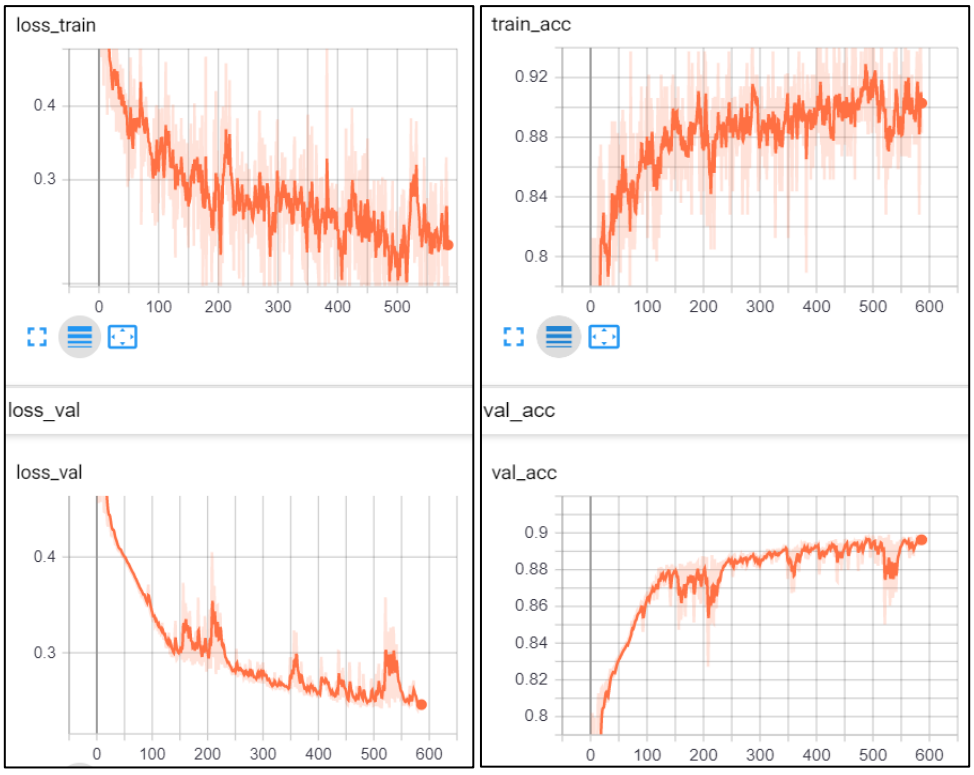

Figure 6: Adam Optimizer performance after parameter tuning

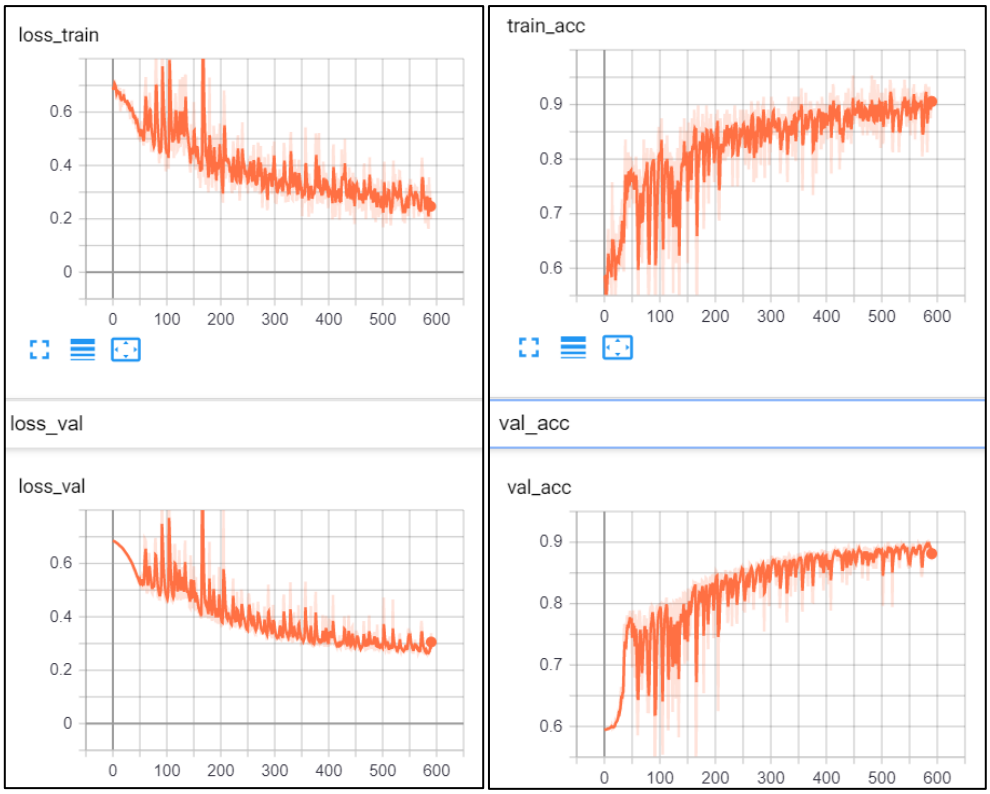

Figure 7: RMSProp Optimizer performance after parameter tuning

References

[1] /@ashwaths. (2019, April 09). How to use the Learning Rate Finder in TensorFlow - Octavian. Retrieved from https://medium.com/octavian-ai/how-to-use-the-learning-rate-finder-in-tensorflow-126210de9489

[2] How to choose CNN Architecture MNIST. (n.d.). Retrieved from https://www.kaggle.com/cdeotte/how-to-choose-cnn-architecture-mnist

[3] /@sidereal. (2018, July 03). CNN Architectures: LeNet, AlexNet, VGG, GoogLeNet, ResNet and more .... Retrieved from https://medium.com/@sidereal/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5

[4] (n.d.). Retrieved from http://cs231n.github.io/convolutional-networks/.

[5] Dont Use Large Mini-Batches, Use Local SGD - arxiv.org. (n.d.). Retrieved from https://arxiv.org/pdf/1808.07217.pdf.