Problem Statement:

Load the CIFAR10 dataset and select a subset of three classes. Split the training set into a training and validation subsets. Vectorize the images and encode the known class labels using categorical encoding. Design a fully connected neural network to perform multi-class classification of this data. Justify your network design decisions (number of hidden layers, number of units per layer, loss function, and evaluation metric). Build and compile the network you designed. Plot training and validation loss as a function of epochs, then plot training and validation accuracy. Tune model hyper parameters to improve performance. Retrain the final model and test performance on the test collection. Report the performance you obtain. Make sure that your program can save and load the weights of the trained network.

We are trying to improve the performance of our model which use the CIFAR 10 data set.

Proposed Solution:

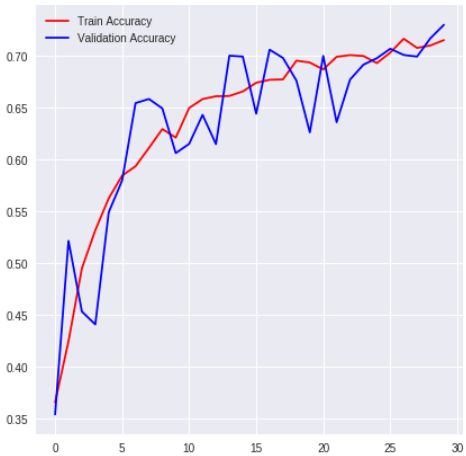
CIFAR_10 Dataset

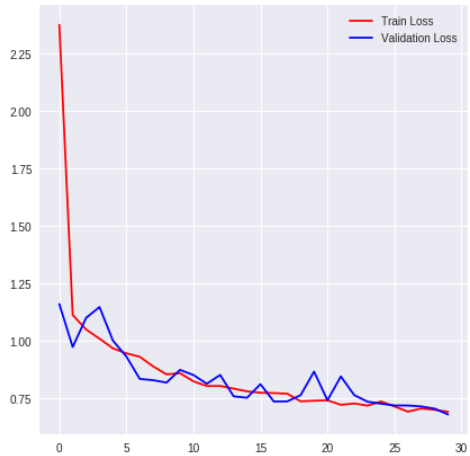
Loading Data, Base Model, Tuning Number of Layers and Size of Hidden Layers , Tuning Learning Rate, Adding Dropout, Training Final Model and Results

1. Keras'sbuilt in CIFAR_10 dataset is being used. The data is loaded using: (x_train_all, y_train_all), (x_test_all, y_test_all) = cifar10.load_data()
2. The Train and Test predictors, with a shape of 32 x 32 x 3 are reshaped into vectors of (32*32*3 = 3072) dimensions.
3. A subset of 3 classes (or a parameterized value of num_classes) is considered for this exercise. o The Train and Test predictors are scaled by dividing the values by 255.
4. The target variable is one-hot encoded to categorical variables.
5. The Train set is further divided into a 70-30 split of Training and Validation data.

Implementation details:

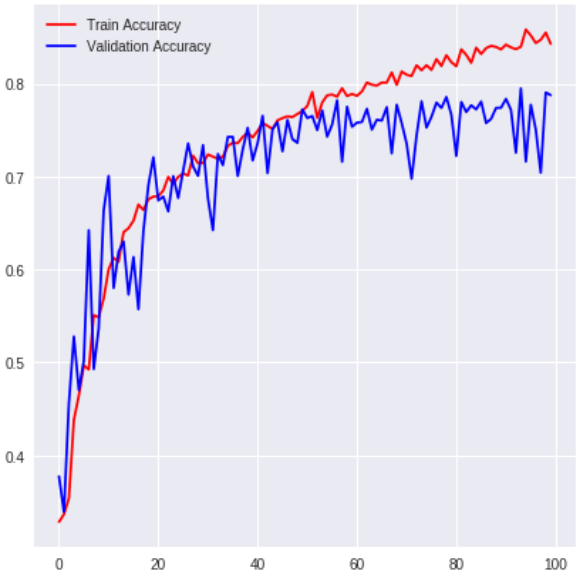
1. Model with 1 Neuron and 64 Neurons was trained as an initial benchmark with gave a validation accuracy of 70%.
2. Architecture: No. of Layers: 1 ; No. of Neurons: 64 Loss: Categorical_crossentropy ; Optimizer: RMSProp ; Metrics: ['accuracy']
3. Activation: Relu for the Hidden Layers, Softmaxfor the Output layer

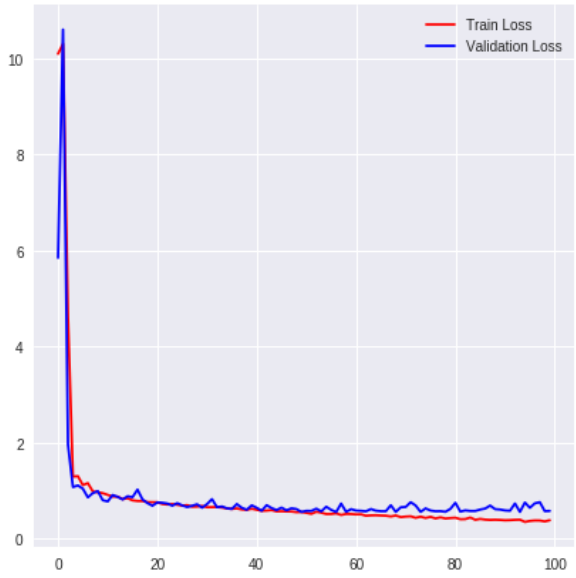




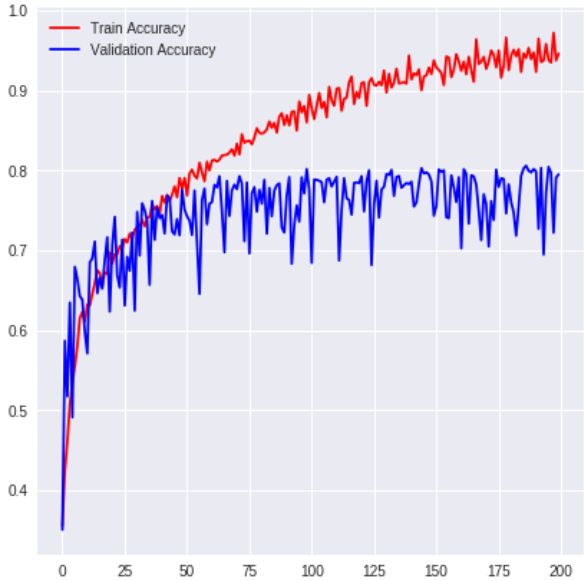
Tuning Layers 4 More Architectures were tested and compared on their validation accuracy

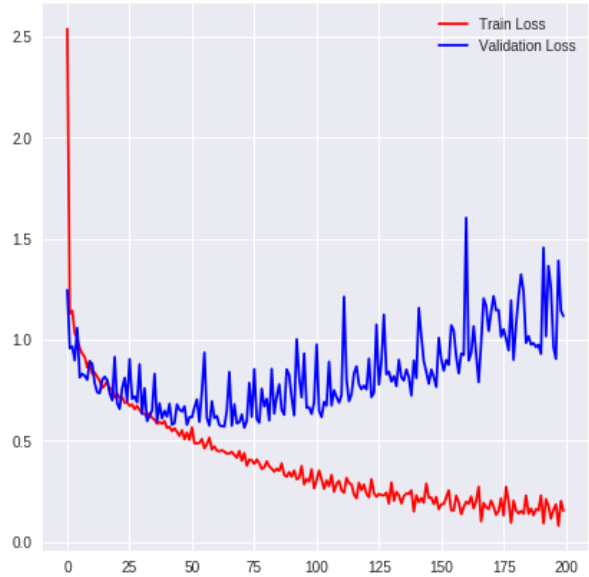
2 Layers – [512, 64] with batch size 512



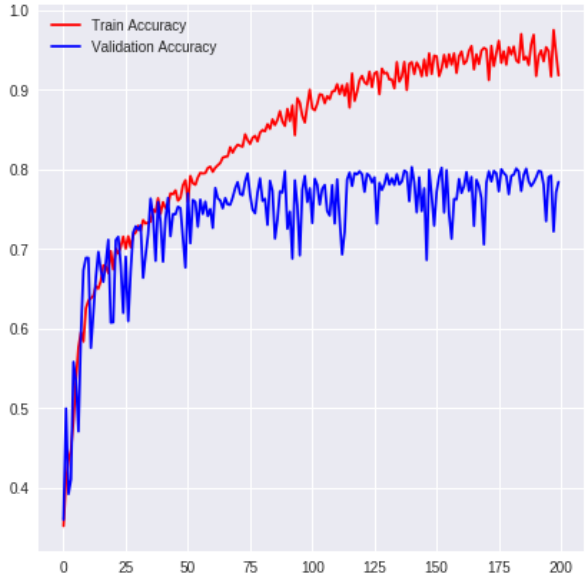


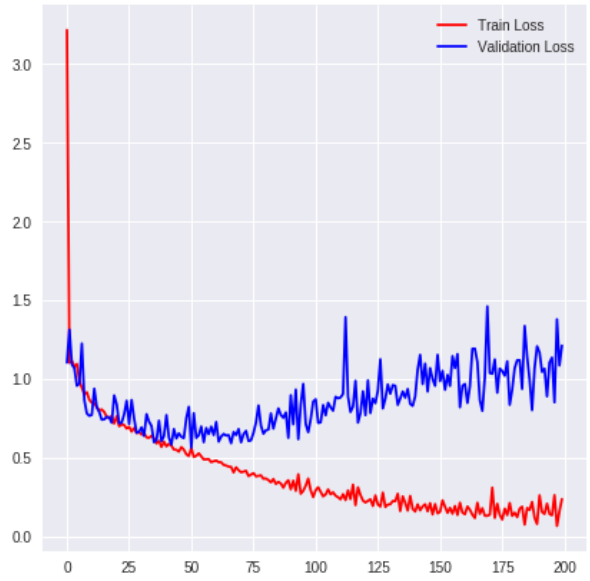
3 Layers –[512, 128, 64] with batch size 512





4 Layers –[512, 256, 128, 64] with batch size: 512



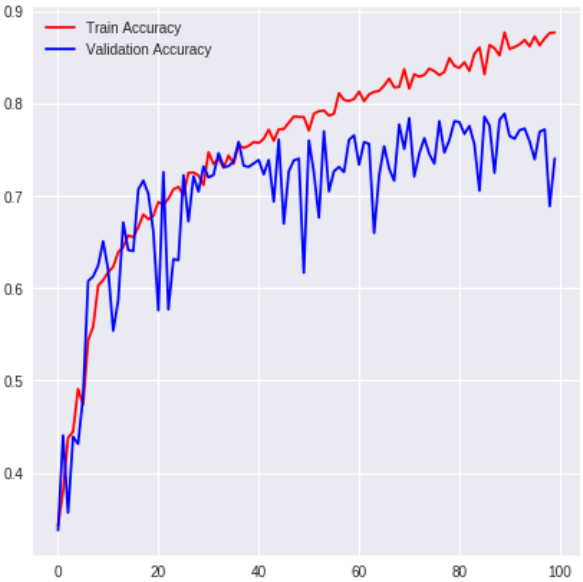


#we can see getting similar result at 3 and 4 layer models, so will go with 3 layers model, with a batch size of 512.

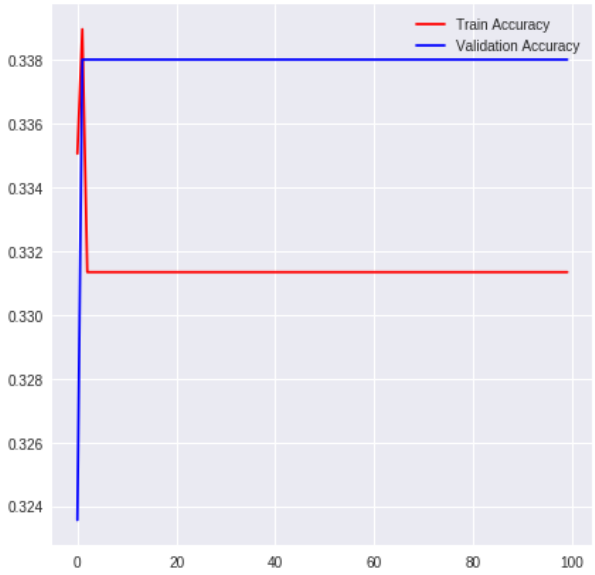
#Epochs - Reducing the epochs to 100, since the Validation Loss indicates overfitting beyond 100 epochs.

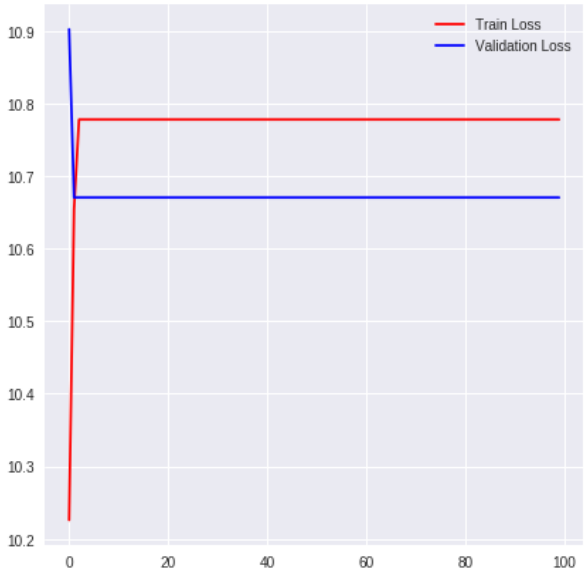
Tuning Learning Rate:

#1 Learning Rate - 0.001

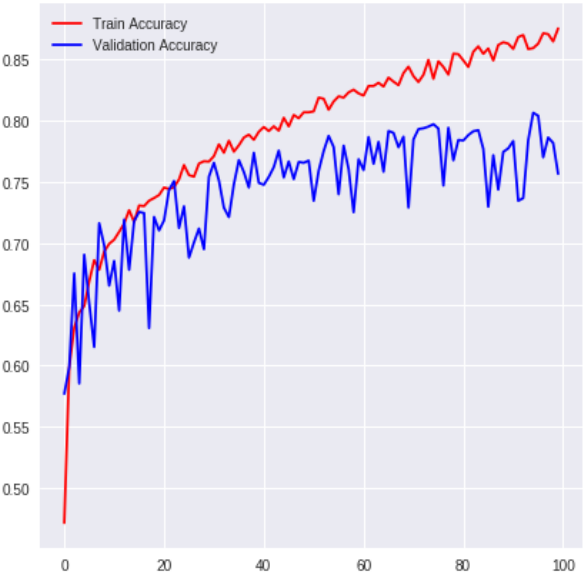


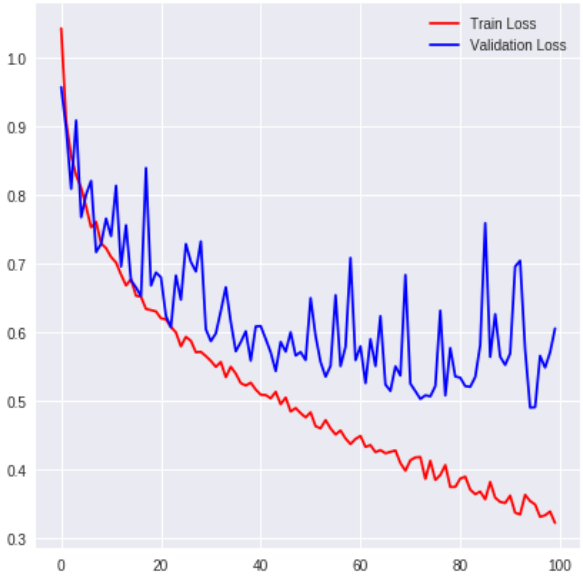
#2 Learning Rate - 0.01





#3 Learning Rate - 0.0001

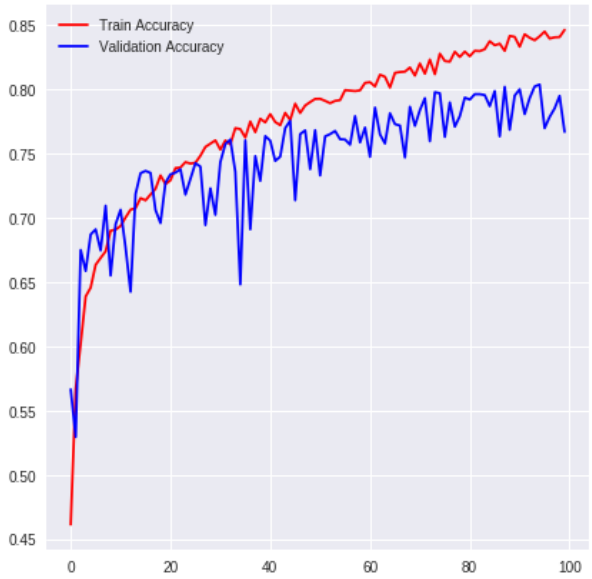


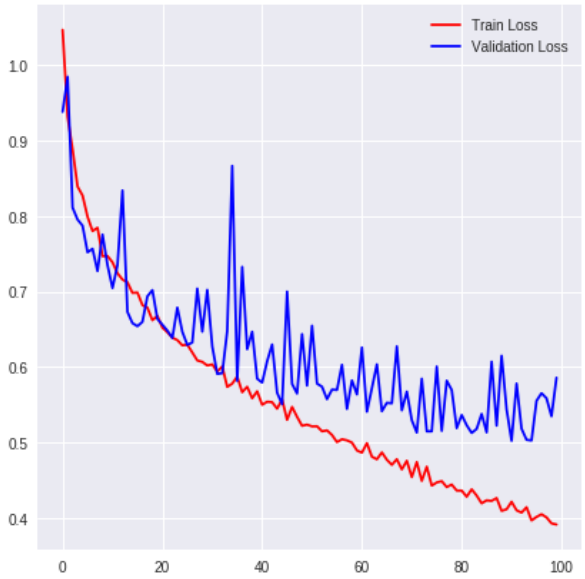


We are getting better Validation Accuracy at 3rd learning rate (0.0001)

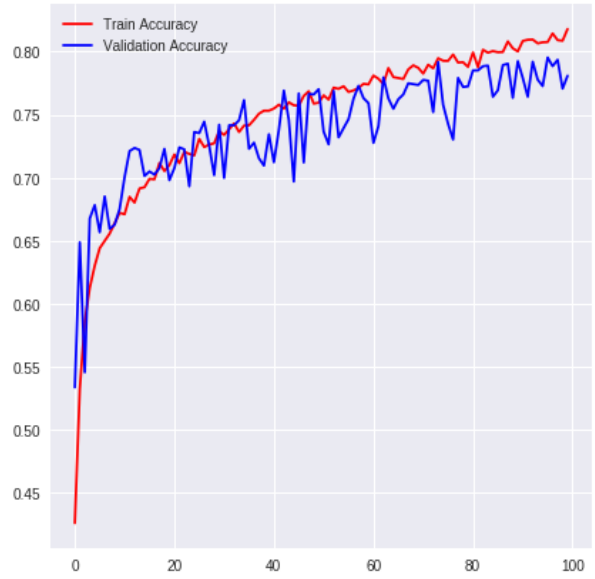
Adding Dropout

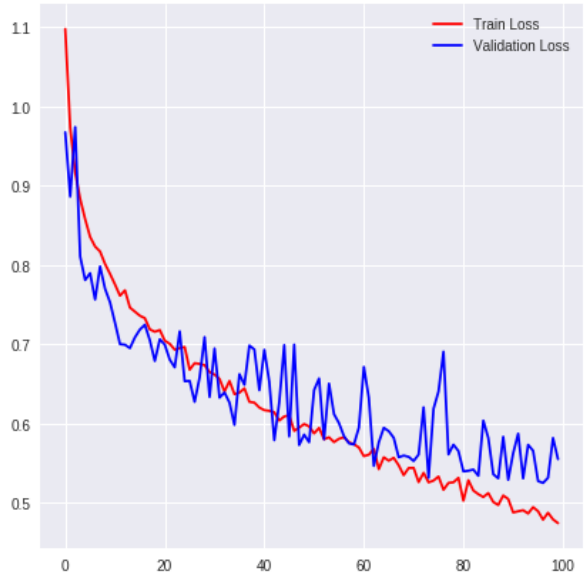
Dropout = 0.2





Dropout = 0.5





Dropout Value of 0.5 fixes overfitting better.

Results and Discussion:

• Number of Layers: 3 • Number of Neurons: [512, 128, 64] • Optimizer: RMSProp • Optimizer Learning Rate: 0.0001 • Dropout: 0.5 • Regularization: NA • Activation Function: Relufor the hidden layers, Softmaxfor the output layer • Loss: categorical_crossentropy • Epochs: 100 • Batch_size: 512

Results -Test Accuracy: **81.53**%

```
epochs = 100
batch_size = 512

final_model = define_model(num_layers=3, num_neurons=[512, 128, 64], input_shape=(x_train_full.shape[1],),
                           optimizer_lr=0.0001, dropout = 0.5)
history = final_model.fit(x_train_full, y_train_full,
                    epochs = epochs,
                    batch_size = batch_size,
                    verbose = 0)

results = final_model.evaluate(x_test, y_test)

results
```

```
3000/3000 [==============================] - 1s 356us/step

[0.48211358865102133, 0.8153333331743876]
```