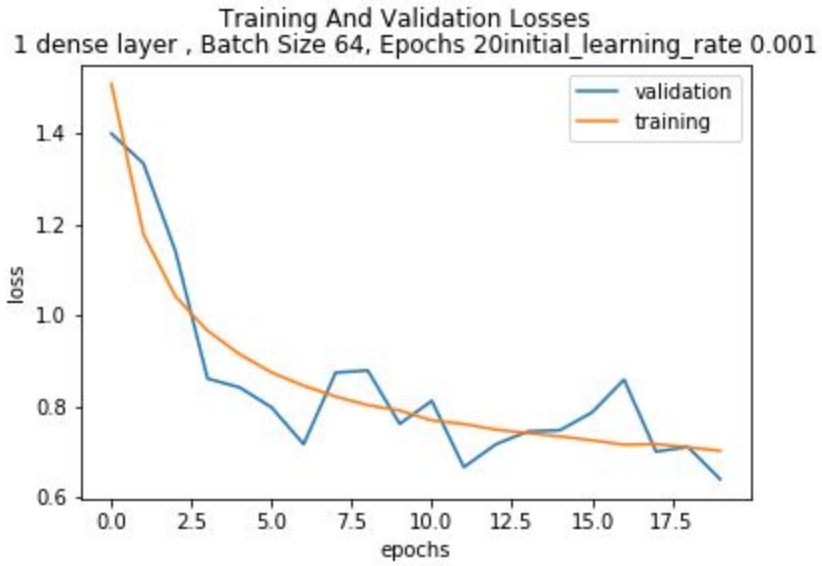
Report Assignment 2(b):
Image Classification With CNN On CIFAR10

Sumit Singh
2016PH10575

Part(A):

1. **Training Loss vs Epoch**:



2. **Best Performing Model**:

   No. of layers and layer type: As specified
   Accuracy: 69.19% (after 10 epochs, absolute part, runs in < 6 min)
   Optimizer used: RMSprop
   Batch Size = 32

3. **Optimizations:**
   - SGD gave a lower accuracy and took higher training times. RMSprop gave the best performance.
   - Adding Dropout layers was not necessary because threshold accuracy (65%) was achieved in merely 10 epochs and there was no significant overfitting in this range
   - Batch Normalization layer was used before softmax layer.
   - Training and Test data was normalized by dividing by 255.0 which led to an improvement in accuracy.

Part (B):

Best Performing Architecture:

Number of layers: 6 CONV2D layers

Best accuracy: 87.3% (after 80 epochs on Google Colab, runs in < 55 min)

Optimizer used: RMSprop

Stride = 1 and 2 in alternate layers

Batch Size: 60

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 32, 32, 48)        1344
_________________________________________________________________
batch_normalization_8 (Batch (None, 32, 32, 48)        192
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 16, 16, 48)        20784
_________________________________________________________________
batch_normalization_9 (Batch (None, 16, 16, 48)        192
_________________________________________________________________
dropout_4 (Dropout)          (None, 16, 16, 48)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 16, 16, 96)        41568
_________________________________________________________________
batch_normalization_10 (Batc (None, 16, 16, 96)        384
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 8, 8, 96)          83040
_________________________________________________________________
batch_normalization_11 (Batc (None, 8, 8, 96)          384
_________________________________________________________________
dropout_5 (Dropout)          (None, 8, 8, 96)          0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 4, 4, 192)         166080
_________________________________________________________________
batch_normalization_12 (Batc (None, 4, 4, 192)         768
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 4, 4, 192)         331968
_________________________________________________________________
batch_normalization_13 (Batc (None, 4, 4, 192)         768
_________________________________________________________________
dropout_6 (Dropout)          (None, 4, 4, 192)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 3072)              0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                30730
_________________________________________________________________
batch_normalization_14 (Batc (None, 10)                40
_________________________________________________________________
activation_2 (Activation)    (None, 10)                0
=================================================================
Total params: 678,242
Trainable params: 676,878
Non-trainable params: 1,364
_________________________________________________________________
None
```

Optimizations Used:

1. **Optimizer**: Stochastic Gradient Descent did not seem to perform well. The best performing optimizer proved to be RMSprop. The optimal starting learning rate for RMSprop was 0.003 and it was decreased by a factor of 2 after every 20 epochs. This was done to provide variable learning rates to RMSprop especially in the later epochs which led to an increase in performance.

```
rmsprop = keras.optimizers.rmsprop(lr= 0.001, decay=1e-7)
```

2. **No. of layers**: Best accuracy was achieved with 6 CONV2D layers. Intermediate CONV2D layers had strides of 2 to speed up the learning per epoch. This led to substantial improvement in running time per epoch (60 sec with stride = 1, 32 sec with stride = 2). Batch Normalization and Dropouts were also added to prevent over-fitting.

```
model.add(Conv2D(48, (3, 3), activation='relu', padding = 'same', strides = 2))
model.add(BatchNormalization())
model.add(Dropout(0.2))
```

3. **Activation:** ReLU was used in all CONV2D layers. Experimentations were done with ELU and Leaky-ReLU and the best performing activation function was ReLU.

4. **Learning Rate**: With Keras learning rate scheduler, it became possible to achieve high accuracy in less number of epochs. However the accuracy did not seem to increase at higher levels. Changing the learning rate of the optimizer after every twenty epochs, however, led to an increase in accuracy by about 2%.

5. **ResNET and other SOTA**: Experiments were done with ResNET to achieve higher accuracy. Even though ResNET was able to achieve high accuracy given a considerable amount of training time, it was not ideal for the present situation because of the training time limit of 1 hour. Improvements to the proposed model helped in achieving higher accuracy than ResNET, given the time limit.

6. **Timeout**: A custom Callback function ‘Timeout’ was used to stop training before 1 hour and return the best model so far

7. **Normalization**: Train and Test data was normalized by dividing it by the maximum pixel value (255.0). Without this the training and test accuracy were severely affected.

8. **ReduceLROnPlateau:** This callback function provided by Keras proved to be effective in accelerating the learning where otherwise the learning would be slow. This function decreases the learning rate whenever there is less change in training accuracy for some ‘patience’ number of epochs, by multiplying the learning rate by some ‘factor’.

```
lr_reducer = keras.callbacks.ReduceLROnPlateau(monitor='acc',
                                               factor=0.5,
                                               patience= 5,
                                               verbose=1,
                                               min_delta=0.01,
                                               min_lr=0.00005,
                                               mode='max')
```

```
Epoch 7/20
833/833 [==============================] - 54s 65ms/step - loss: 0.4659 - acc: 0.8392
Epoch 8/20
833/833 [==============================] - 54s 65ms/step - loss: 0.4636 - acc: 0.8376
Epoch 9/20
833/833 [==============================] - 54s 65ms/step - loss: 0.4620 - acc: 0.8402

Epoch 00009: ReduceLROnPlateau reducing learning rate to 0.0010000000474974513.
Epoch 10/20
833/833 [==============================] - 54s 65ms/step - loss: 0.4374 - acc: 0.8483
```

9. **Data Augmentation**: Real-time data augmentation was done with Keras ImageDataGenerator class to improve the performance of the model on unseen data.In every batch, images were randomly shifted horizontally and vertically, were given a random rotation between 0 and 15 degrees, and were randomly flipped horizontally. This led to significant increase in test accuracy at the cost of longer training time.

```
datagen = ImageDataGenerator(rotation_range= 15,
                             width_shift_range=0.1,
                             height_shift_range = 0.1,
                             horizontal_flip=True)
datagen.fit(X_train)


rmsprop = keras.optimizers.rmsprop(lr= 0.003, decay=1e-7)
model.compile(loss='categorical_crossentropy', optimizer= rmsprop, metrics = ['accuracy'])
h = model.fit_generator(datagen.flow(X_train, Y_train, batch_size= batch_size),
                        steps_per_epoch= X_train.shape[0]//batch_size,
                        epochs= 20,
                        workers = 4,
                        callbacks = [lr_reducer, end_training],
                        shuffle = True)
```