# Classification of CIFAR-10 database using CNN

Previously we used data augmentation with more CNN layers to achieve efficiency of 84%.

This time we compared both GridSearchCV and RandomizedSearchCV to find the best way of tuning the parameters.

We found out that RandomizedSearchCV is more efficient in searching the best parameters instead of trying out all the parameters [1].

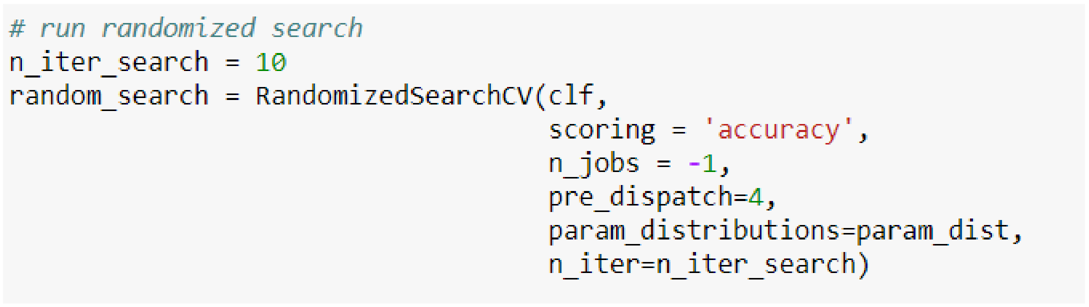
Then we started implementing the RandomizedSearchCV we found out that it doesn't work with data augmentation and even if it's does it's not at all efficient with it.

So, we had to disable Data augmentation and remove the ImageDataGenerator object.

Then we tried out different parameters with RandomizedSearchCV but it was crashing in the end for some reason instead of plotting the results and opting a best set of hyperparameters.

Then we tried a way to change optimizers by passing it in param_dict in RandomizedSearchCV but it was not possible so we had to change it manually.

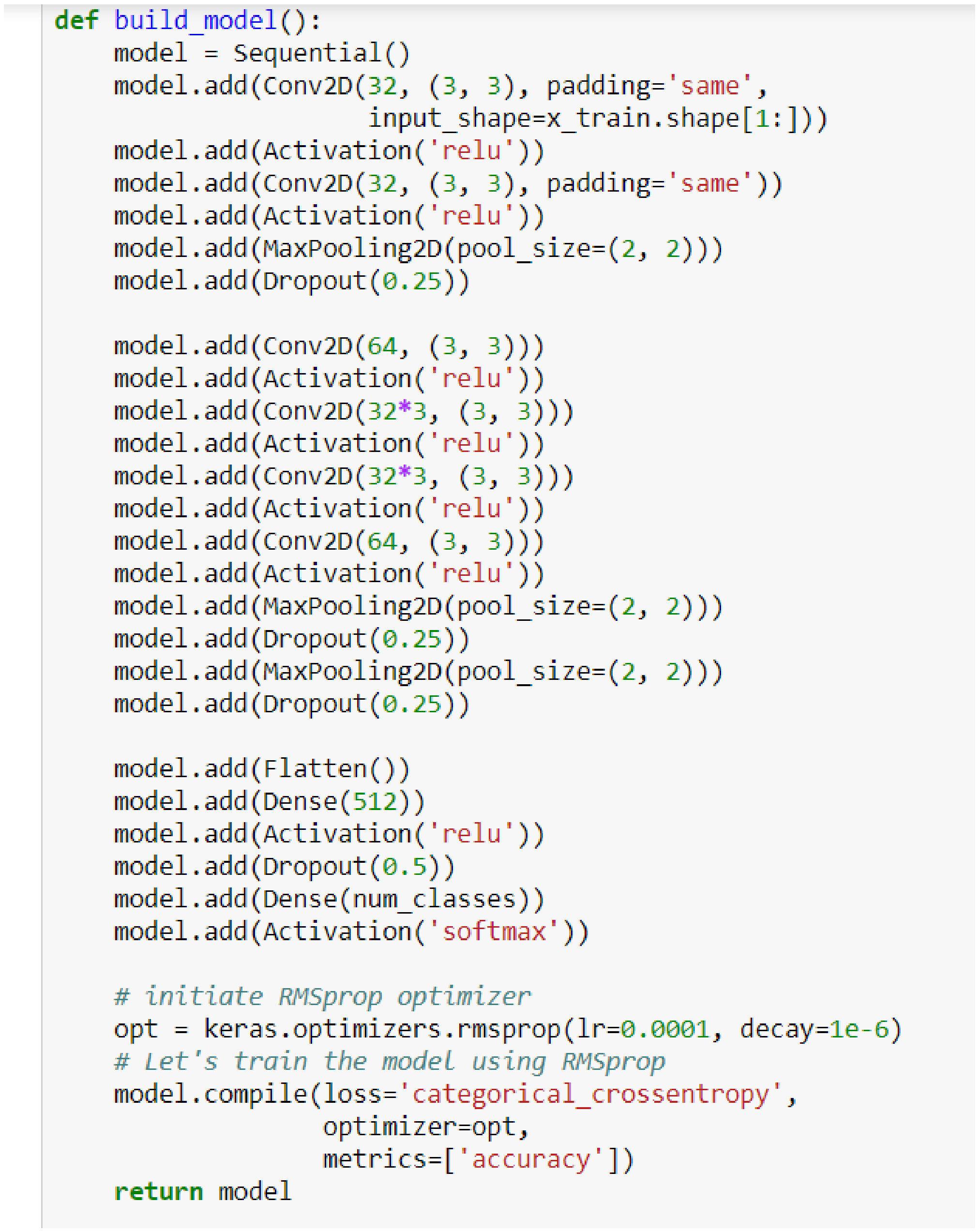
The default RmsProp turned out to be highly efficient.



```
# run randomized search
n_iter_search = 10
random_search = RandomizedSearchCV(clf,
                                   scoring = 'accuracy',
                                   n_jobs = -1,
                                   pre_dispatch=4,
                                   param_distributions=param_dist,
                                   n_iter=n_iter_search)
```

We tried to optimize our resource by setting n_jobs but memory was getting filled very quick so we used pre_dispatch to control spawning of jobs and control efficiency. RandomizedSearchCV takes number of search within which we'd like to tune our parameter. We set it to 10 so that it'll run the experiment 10 times with different random set of parameters and return the best possible solution.

Link to GitHub: [https://github.com/truptigore17/Cognitive-Computing](https://github.com/truptigore17/Cognitive-Computing)
**YouTube link:** [https://youtu.be/sebgpiOwvQQ](https://youtu.be/sebgpiOwvQQ)

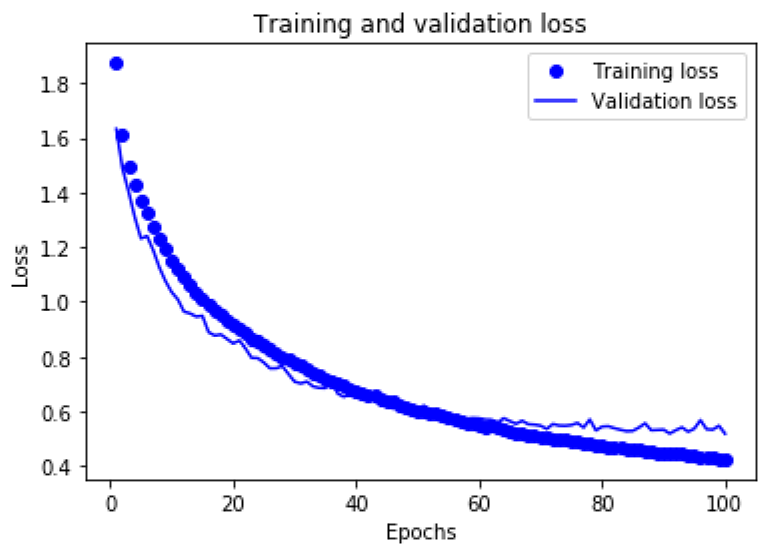


```
def build_model():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), padding='same',
                     input_shape=x_train.shape[1:]))
    model.add(Activation('relu'))
    model.add(Conv2D(32, (3, 3), padding='same'))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Conv2D(64, (3, 3)))
    model.add(Activation('relu'))
    model.add(Conv2D(32*3, (3, 3)))
    model.add(Activation('relu'))
    model.add(Conv2D(32*3, (3, 3)))
    model.add(Activation('relu'))
    model.add(Conv2D(64, (3, 3)))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Flatten())
    model.add(Dense(512))
    model.add(Activation('relu'))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes))
    model.add(Activation('softmax'))

    # initiate RMSprop optimizer
    opt = keras.optimizers.rmsprop(lr=0.0001, decay=1e-6)
    # Let's train the model using RMSprop
    model.compile(loss='categorical_crossentropy',
                  optimizer=opt,
                  metrics=['accuracy'])
    return model
```

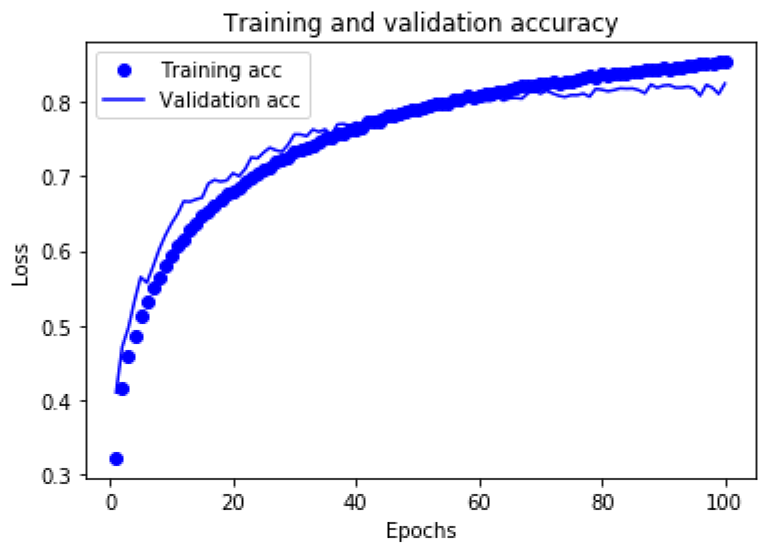
Above is our build function which'll build and compile the model each time and pass it to the search.

Link to GitHub: https://github.com/truptigore17/Cognitive-Computing
**YouTube link:** https://youtu.be/sebgpiOwvQQ

At last the model yielded following results



At last as we can see, the validation loss is more than the training loss so the model started overfitting.



As we can see, the model reached the maximum of 82% accuracy.

Link to GitHub: https://github.com/truptigore17/Cognitive-Computing
**YouTube link:** https://youtu.be/sebgpiOwvQQ

References

http://www.arunprakash.org/2017/04/gridsearchcv-vs-randomizedsearchcv-for.html

Link to GitHub: https://github.com/truptigore17/Cognitive-Computing
**YouTube link:** https://youtu.be/sebgpiOwvQQ