Lab02 Report | Submission | Deep Learning and Neural Nets

Submitted By: Anusha Basavaraja
(anba9017@colorado.edu)

Part 1 – Influence of Regularization:

Method:

Dataset – I am using CIFAR10 image dataset available in Keras. It has 10 classes with inbuilt train and test split in the dataset (50,000 training images with corresponding labels and 10,000 test images with corresponding labels respectively). The 10 classes in the labels are denoted numerically in the range 0-9 which are converted into categorical 1 hot encoding during data preprocessing stage before feeding this into the model. Since the Keras CIFAR10 dataset had inbuilt train-test split, I took the liberty of splitting the available training dataset into train-validation split with 40,000 and 10,000 images respectively with corresponding labels. I made the validation dataset to have 10,000 examples because the test split originally contains 10,000 examples. So, I wanted to keep the size of validation and test set equal.

Experiment – In order to experiment with 8 different CNN models, I have chosen two set of models.

1st set: There are 4 models with 2 convolution layers – {Base model 1 with no Regularization, Adding L2 regularization on Base model 1, Adding Dropout on Base model 1, Adding Batch Normalization on Base model 1}. Base model 1 network layers look like – input – conv layer 1 – maxpool – conv layer 2 – maxpool – flatten – dense – output.

2nd set: There are 4 models with 3 convolution layers – {Base model 2 with no Regularization, Adding L2 regularization on Base model 2, Adding Dropout on Base model 2, Adding Batch Normalization on Base model 2}. Base model 2 network layers look like – input – conv layer 1 – maxpool – conv layer 2 – maxpool – conv layer 3 – maxpool –flatten – dense 1 – dense 2 – output.

For all these 8 models, the following hyperparameters are kept constant explicitly throughout with the following values:

- No. of iterations for training = 30 epochs
- Loss = Cross entropy
- Batch size = 64
- Optimizer = Adam
- Activation function at each layer = ‘ReLu’ & ‘softmax’ at the output layer.
- No. of filters at each convolution layer = 64
- Size of the filters = 3*3
- Maxpooling after each Convolution layer with pool size = 2*2
- Each dense layer has 100 nodes
- Output layer (a dense layer) has 10 nodes for 10 output classes.
- Regularization – L2 has 0.00001 and Dropout has 0.1 values as arguments.

I chose these explicit values based on the observations from coding tutorials and after going through some generic articles from the web as it looks like standard or close to standard values. Rest of the hyperparameters are kept untouched with whatever available by default in the library. The models are executed on Google Colab by changing the runtime to GPU.

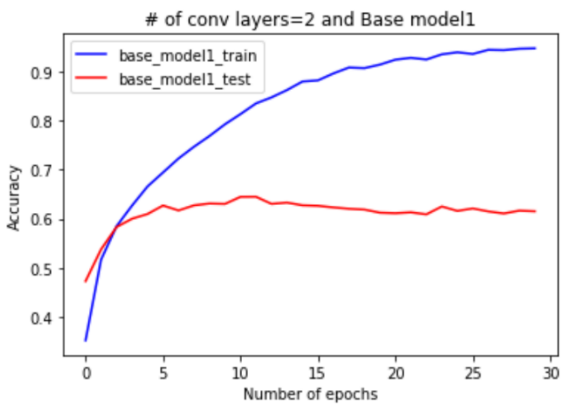
Results:

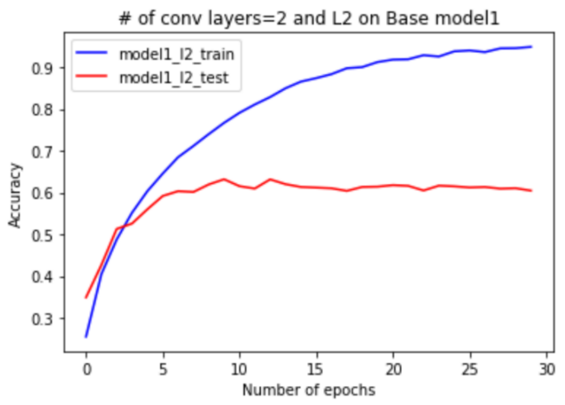
After execution of 8 models, the results obtained from each of the model executions are as follows:

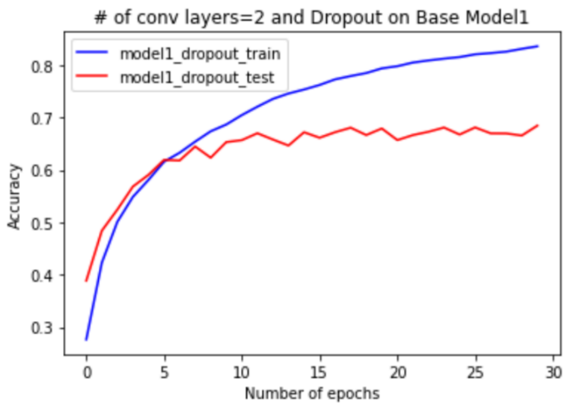
Sl No.	*Model*	*No. of Conv Layers*	*Time taken for Training*	*Best Training Accuracy*	*Best Validation/Test Accuracy*
1.	Base model 1 (No Regularization)	2	2min 22sec	94.64%	64.47%
2.	L2 on Base model 1	2	1min 41sec	94.82%	63.12%
3.	Dropout on Base model 1	2	2min 22sec	82.07%	68.45%
4.	Batch Normalization on Base model 1	2	2min 23sec	98.18%	70.38%
5.	Base model 2 (No Regularization)	3	2min 22sec	86.86%	66.91%

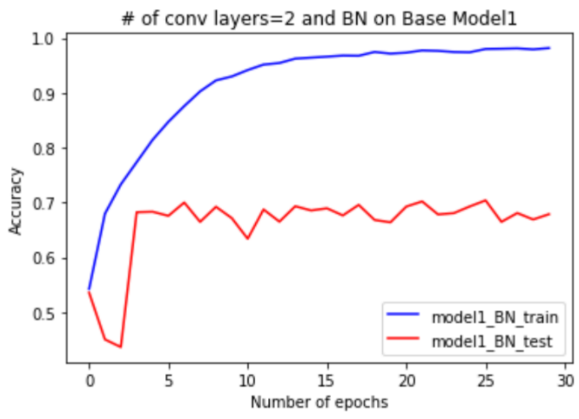
6.	L2 on base model 2	3	1min 58sec	90.28%	68.80%
7.	Dropout on Base model 2	3	1min 45sec	73.93%	71.61%
8.	Batch Normalization on Base model 2	3	1min 48sec	95.02%	73.02%

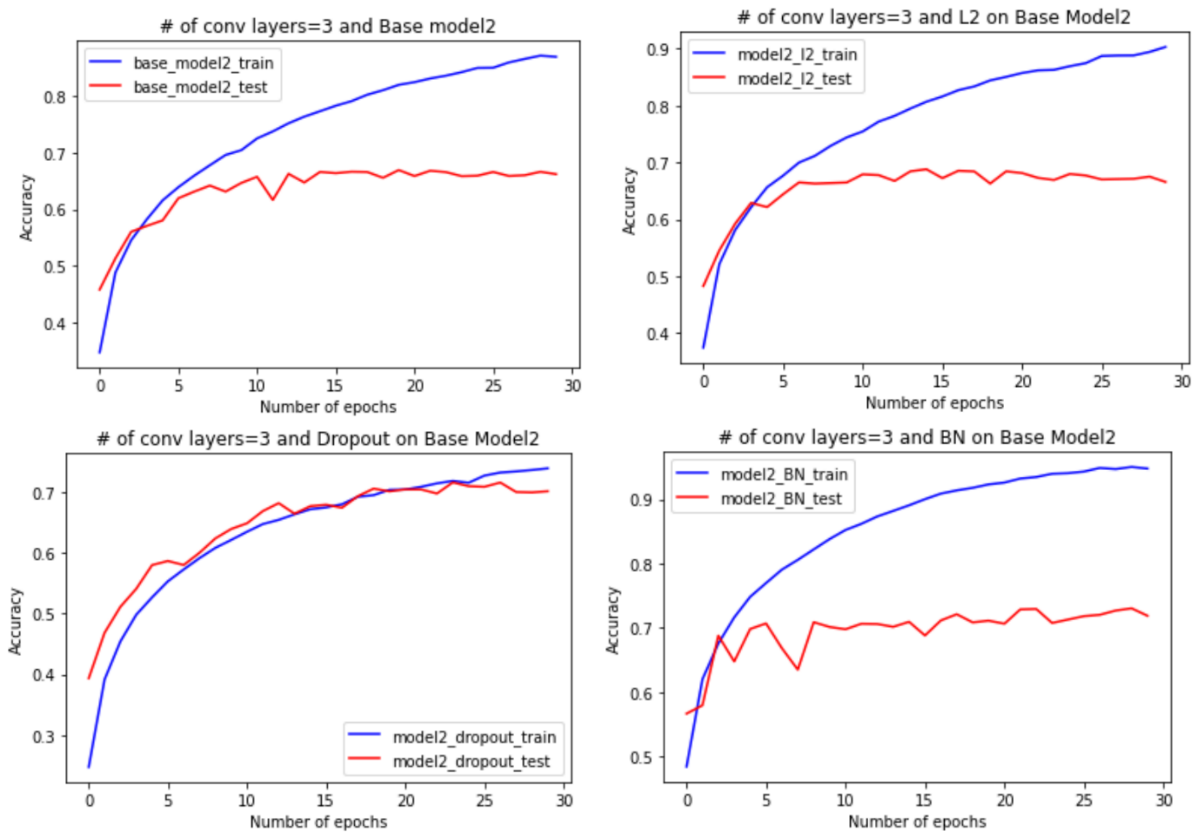
***Table 1**: Results from executing 8 models with different no. of CNN layers and Regularizations methods.*











***<u>Fig</u>*: **1)** 2 Conv layers with no regularization, **2)** 2 Conv layers with L2, **3)** 2 Conv layers with Dropout, **4)** 2 Conv layers with Batch Normalization, **5)** 3 Conv layers with no regularization, **6)** 3 Conv layers with L2, **7)** 3 Conv layers with Dropout, **8)** 3 Conv layers with Batch Normalization. All 8 models executed on the dataset with 40,000 images and 10,000 images in train-validation splits. Blue curve represents the training accuracies on Training dataset and whereas Red curve represents the testing accuracies on Validation dataset.*

Even though Model 8 (adding Batch Normalization on base model 2 with 3 convolution layers) looks like better performing model based on the accuracy score, if we consider the fact of consistency, Model 7 (adding Dropout on base model 2 with 3 convolution layers) has good consistent results when we compare model performance on both training and validation datasets. So, instead of choosing one final model, I planned to experiment and run the models twice by choosing two outcomes from the results observed in the above table. Final model 1 (Dropout with 3 convolution layers i.e., Model no. 7) and Final model 2 (Batch Normalization with 3 convolution layers i.e., Model no. 8) – both executed on train-test datasets instead earlier train-validation splits with more data now available for training in each epoch. The results of executing the Final models 1 & 2 are shown below.

Sl No.	*Model*	*No. of Conv Layers*	*Time taken for Training*	*Best Training Accuracy*	*Best Validation/Test Accuracy*
1.	Final Model 1 (Adding Dropout to Base Model 2 with 3 convolution layers)	3	2min 15sec	74.31%	73.61%
2.	Final Model 2 (Adding Batch Normalization to Base Model 2 with 3 convolution layers)	3	2min 55sec	94.66%	73.53%

Table 2: *Final Models 1 & 2 and their respective outcomes*

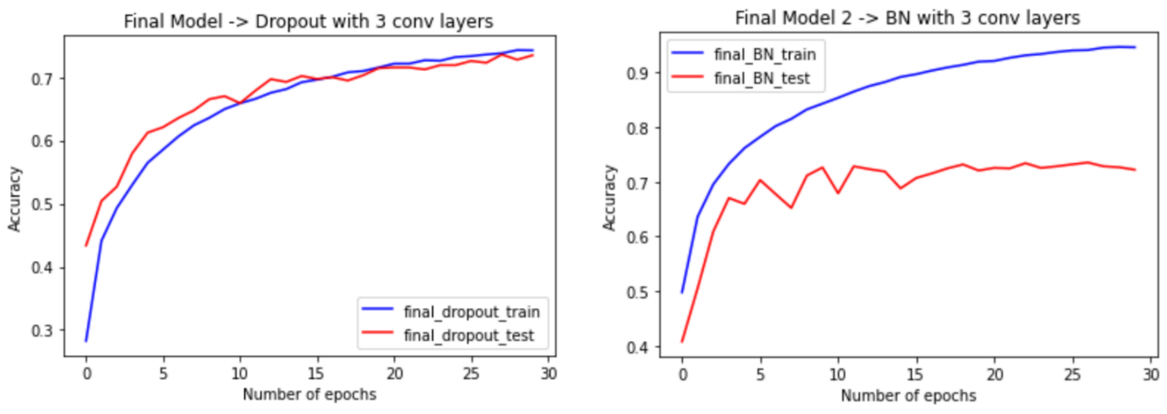


Fig: ***1)*** *Final Model 1 with 3 Conv layers with Dropout,* ***2)*** *Final Model 2 with 3 Conv layers with Batch Normalization. Both final models executed on the dataset consisting of 50,000 training images and 10,000 testing images. Blue curve represents the training accuracies on Training dataset and whereas Red curve represents the testing accuracies on Test dataset.*

From the above table and figures, we can arrive at the conclusion that, Final Model 1 is better than Final Model 2 because in my opinion, the ideal model is the model that is consistent throughout i.e., during training as well as testing phase of model execution whereas Final Model 2 performs really well during training but fails to maintain that consistency during testing phase thereby resulting in the huge accuracy drop. This is clearly sign of overfitting. Final Model 1 performs well with consistent performance thereby showing some spike in the accuracy when comparing model executions between train-validation splits and train-test splits. This is because during train-validation splits, the size of the dataset used is 40,000-10,000 images and train-test splits, the size of the dataset used is 50,000-10,000 images. This is an example of data augmentation to the training dataset which preserves the consistent performance of the model during training and testing phase along with slightly improving the accuracies in the Final Model 1 thereby allowing the model to understand the relationship and correlation between the inputs and the outputs. We can notice that in the graph for Final Model 1, both learning curves of training and testing overlaps with each other which is the ideal case for consistent performance from a model whereas in case of Final Model 2, there is huge gap between the learning curves of training and testing thereby showcasing the scenario of overfitting.

<u>Analysis</u>:

1) *What is observed when comparing the use of regularization techniques to not using any regularization during training?*

   Regularization is used to reduce the test errors intended to reduce the generalization errors thereby modifying the learning algorithm. Also, regularization is one of the best ways to avoid overfitting observed in a model. If we observe the results from Table 1 and learning curves from 8 CNN models, we can notice that there is a huge gap between the train and test learning curves which shows that it is an overfitting scenario. But, if we add regularization during the model training phase, we can observe that there is slight to substantial reduction in the gap between the train & test learning curves depending on the type of regularization incorporated. My speculation to this observation is that with neural networks having lots of model parameters introduces more complexities into the model thereby also learning the noise available within the dataset. But in case regularization, the aim of the model now is to generalize the model predictions such that model performance is consistent thereby reducing overfitting. This is achieved by different regularization techniques. But all regularization techniques tend to reduce overfitting by penalizing or adding a constraint to the loss function, add some simplicity into the model on top of the existing model, add more training examples, stop the training early and so on.

2) *Did certain regularization technique and/or network depth (different convolution layers) lead to consistently better results?*

Yes, especially Dropout regularization technique has consistent performance in case of regularization and model with 3 convolution layers has better performance in comparison with other models with 2 convolution layers. When considering both regularization and network depth, the dropout model with 3 convolution layers has good consistent performance which also kind of tops the list. This might be because, the dataset used might demand for more complexity in the network (so adding 3 convolution layers) but yet requires some simplicity (dropout – randomly drop the neurons in the layers) for reducing the gap between the learning curves for training and testing in order to have consistent results during training, validation and testing phases.

3) *Insights gained by looking at the learning curves?*

In the normal scenario with the models with no regularization, with the more no. of training epochs, the model really learns well by giving accuracies over 90-95% during training and 65-70% during validation/testing. It is clearly overfitting scenario. But with regularization techniques, even though there are some slight ups and downs in the testing accuracies, they do their job in reducing the gap between the learning curved obtained for training and validation/testing. But, I did not notice any underfitting scenario while executing any of the models. In order to declare a model as better performing one, the ideal case is the one where the model is not surprised by the unseen data i.e., model should have acceptable and consistent performance during training and testing phases. If a model performs really well on training dataset by also learning the noise but fails to generalize when predicting on the unseen test set, then we should understand that the model is not performing well (overfitting scenario). So, regularization techniques help to scale the gap down a bit between the train and test learning curves so that model can generalize and not get surprised when predicting on test data in the real-time.

4) *Trade-offs between training time and choices for no. of convolution layers and regularization techniques?*

With sufficient no. of experiments, we can observe that by adding more convolution layers to the model with regularization, the model performs well even trained for shorter duration. But, with not sufficient no. of convolution layers, with or without regularization, even though the model is trained for longer, the model might have acceptable accuracies during training but the performance on the validation/test dataset is not good enough showing substantial fall in the accuracies. This might be because, the dataset that I am using for this lab demands for good complexity in the network in order to extract the distinctive features for good predictions as we are now dealing with images dataset which is more complex compared to just the plain numeric digits/values which served as features in simple binary classification tasks worked previously with.

5) Performance comparison for top-performing model configuration when tested on validation and test dataset?

As per my opinion, I would like to mention that Model no. 7 (Final Model 1) is the top-performing model when experimented with all 8 CNN models. This model not only has consistent results when testing on validation set, but it maintains the same consistency even when working with the test dataset. It also must be noted that, the when testing on the test dataset (10,000 images) after training on both training and validation datasets (includes 50,000 images altogether), there was in fact slight increase in the performance. The testing on validation data resulted in 71.61% accuracy and testing on test dataset resulted in 73.61%. This is due to addition of more data into training dataset. Before testing on validation dataset, the model was trained on dataset containing 40,000 training examples whereas before testing the model on the test dataset, the model was trained on the dataset containing 50,000 training examples. Looks like the addition of 10,000 images into the training dataset played a very important in improving the model efficiency not only in terms of accuracy but also in terms on consistency. This proves that having huge datasets with sufficient training examples, helps the model to overcome overfitting thereby improving the model efficiency, consistency and performance. If we closely observe the learning curves of Model no. 7 (testing on validation dataset) and Final Model 1 (testing on test dataset), even though the red curve is almost close to blue curve in Model no.7, it starts deviating at the end thereby showing the fall in the accuracy. But if we observe the red curve in Final Model 1, it does not deviate as much as found in Model no. 7. This finally helps us to understand that adding more data into training dataset, really helps the model to perform better.

<u>**Part 2 – Interpreting CNN Representations**</u>:

Method:

<u>*Dataset*</u> – I am using CIFAR10 image dataset available in Keras. It has 10 classes with inbuilt train and test split in the dataset (50,000 training images with corresponding labels and 10,000 test images with corresponding labels respectively). The 10 classes in the labels are denoted numerically in the range 0-9 which are converted into categorical 1 hot encoding during data preprocessing stage before feeding this into the model. Since the Keras CIFAR10 dataset had inbuilt train-test split, used the same for this part 2 of the lab. Used one of the images in test set (very first image – 'cat') to visualize the feature maps at the three different convolution layers of the model finalized.

<u>Experiment</u> – I have chosen to visualize and analyze the feature maps at three different convolution layers from the Final model. The final model chosen from Part 1 of this lab is Final Model 1 (Dropout with 3 convolution layers) which has the following network layer architecture: input – conv layer 1 – maxpool – conv layer 2 – maxpool – conv layer 3 – maxpool – flatten – dense – dense – output. The model has following hyperparameters that are kept constant explicitly throughout with the values given below:

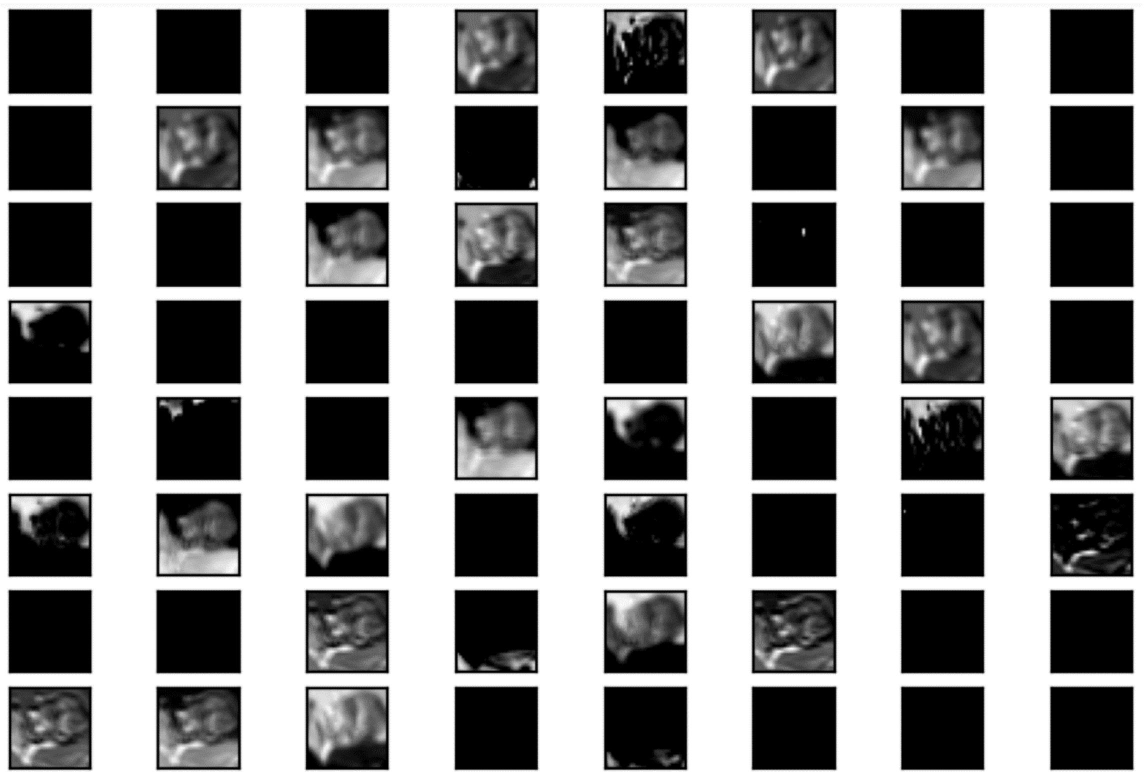
- No. of iterations for training = 30 epochs
- Loss = Cross entropy
- Batch size = 64
- Optimizer = Adam
- Activation function at each layer = 'ReLu' & 'softmax' at the output layer.
- No. of filters at each convolution layer = 64
- Size of the filters = 3*3

- Maxpooling after each Convolution layer with pool size = 2*2
- Each dense layer has 100 nodes.
- Dropout layer as 0.1 value as argument.
- Output layer (a dense layer) has 10 nodes for 10 output classes.

I chose these explicit values based on the observations from coding tutorials and after going through some generic articles from the web as it looks like standard or close to standard values. Rest of the hyperparameters are kept untouched with whatever available by default in the library. This experiment was executed on Google Colab by changing the runtime to GPU.

<u>Results</u>:

There are 64 feature maps available at each convolution layer of Final Model 1 which has 3 convolution layers in its architecture. I was able to plot all 64 feature maps resulted from 64 filters at each convolution layer using an image labelled as 'cat'. Since there are 3 convolution layers in the final model, there are a total of 64 * 3 = 192 feature maps obtained from 3 convolution layers visualized for an image of cat.



<u>Fig</u>: *64 feature maps obtained from 64 filters at the very 1st convolution layer.*

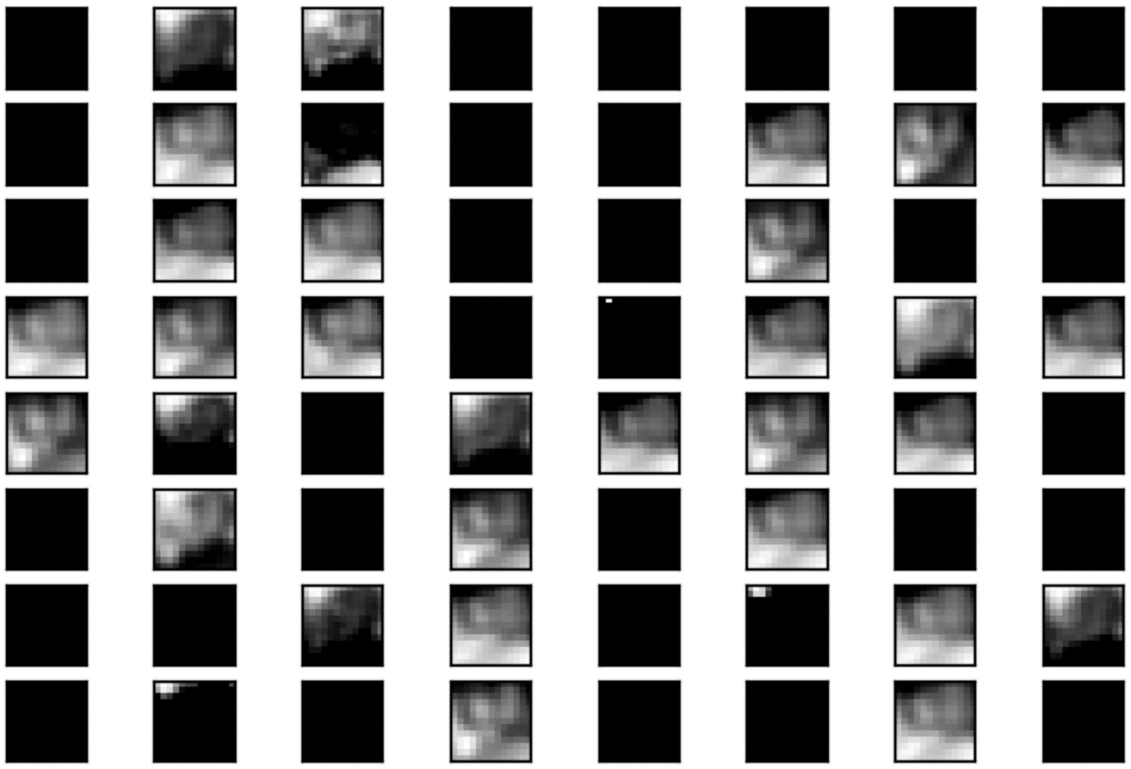


Fig: *64 feature maps obtained from 64 filters at the 2nd convolution layer.*

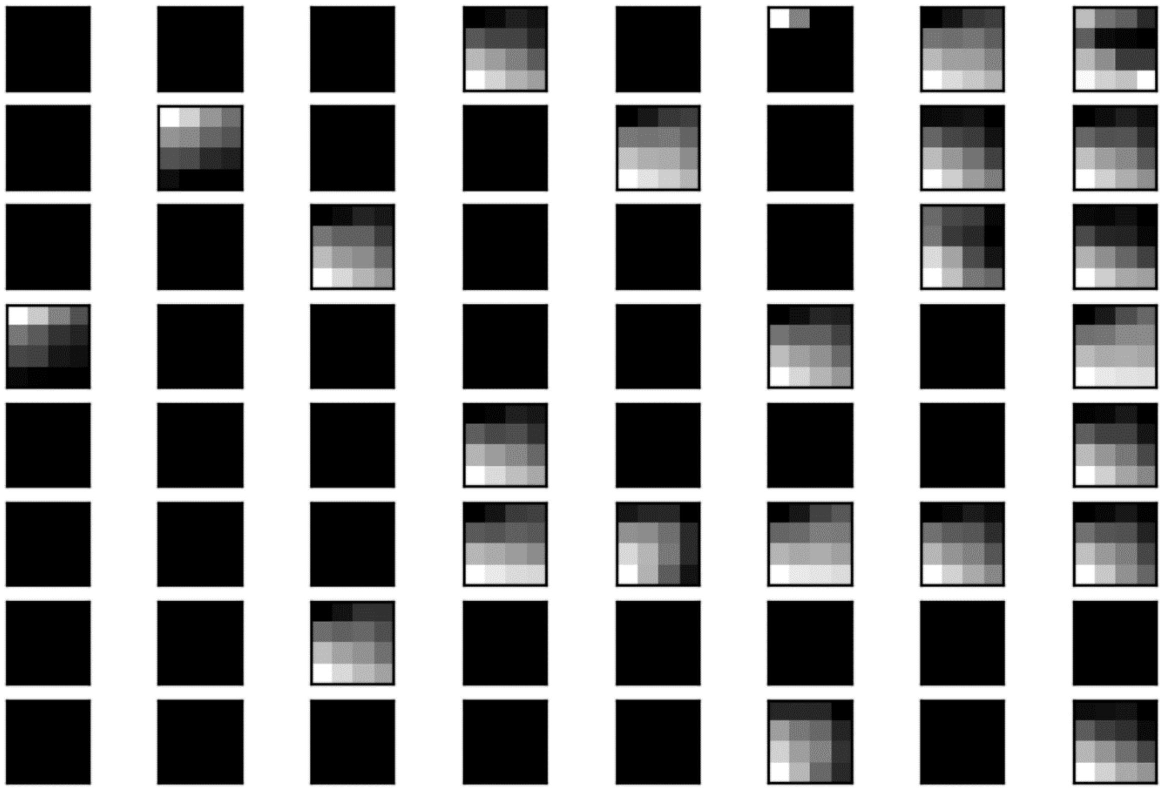


Fig: *64 feature maps obtained from 64 filters at the 3rd convolution layer.*

Note: Two ways of model instantiation used for my lab experiments. 1) Call Sequential() first, then add the model layers. 2) Pass the model layers within the Sequential() call. I had to use 1st type of model instantiation for Part 1 experiments (Regularization) as 2nd type did not perform well with respect to model results. But for Part 2 experiments (Visualizing feature maps), I had to use 2nd type for model instantiation as 1st type did not go through and constantly resulted in error while executing it. After talking to TA, they suggested to proceed with 1st type of model instantiation for Part 1 experiments and 2nd type of model instantiation for Part 2 experiments.

Analysis:

1) *What can you infer about what the model learned?*

   I believe that at each layer, there is different levels of learning that happens in a CNN model. Now that we are dealing with the image data (2D) in CNN model, distinctive features will be accumulated and learned over the course of layers in order to extract good level of information so that the model can predict better. In case of CNN models, it is important to have sufficient number of convolutional layers to extract different levels of information as features from the images data as the learning happens through these layers in contrast to fully connected layers that are used in case of simple classification tasks dealing with straight forward features (1D) extracted from simple datasets. When dealing with 1D data in case of dense layers, even though the network is not complex, if we train the model for much longer, it would help to get better results. But, in case of CNN models used for image classifications (2D data), having good number of convolutional layers is a must as learning distinctive features from the image happens through the course of these layers where each layer concentrates on extracting specific set of information on these images. In CNN models, if there are not sufficient convolutional layers, even though if we train the model for very long, the prediction results will not be as good as those obtained from those models that has good number of convolutional layers.

2) How does what is learned differ at different layers of the network?

   I was able to plot all 64 feature maps that are resulted from 64 filters at each convolutional layer. There are 3 different convolutional layers and the feature maps from all three layers are shown under results section. After analyzing the feature maps from these 1st, 2nd and 3rd convolutional layers, I did read an article available at the link – https://machinelearningmastery.com/how-to-visualize-filters-and-feature-maps-in-convolutional-neural-networks/ to understand how the image patterns are learned in CNN networks. Looks like the feature maps from the 1st convolutional layer has more fine-grained information on the images by focusing more on the centric image where we can notice the cat. Going forward, the feature maps from the 2nd convolutional layer has some distorted info on the focus image of cat itself but here it looks like there is some blur. If we now observe the feature maps from 3rd convolutional layer, we can now notice that the focus is more into generic details of the image as we no longer find any existence of anything that looks like a cat. From this, we can come to conclusion that as we are close to the input layer, the feature maps show the finer details with respect to the images highlighting and focusing more on the fine lines, centric objects, foreground, background and so on. Moving forward towards the output layer, it looks like the feature maps show more generic information and details from the images which almost looks like visualization of filters with more white colored pixels showing the focus of weights that passes the info and pitch-dark black pixels showing almost no focus. This might be because, as we are close to the input layer, there is more information available on the images which allows the filters to extract and focus on fine-grained details available. But as we move far and far from the input layer towards the output layer, since we also use pooling after each convolutional layer, the size of the info in the form of matrix shrinks moving away from the input by using the filters, which kind of makes the finer information to be lost with each of the pixel value vanishing. So, it results in more of generic info rather than specific ones which almost looks like filters from that convolutional layer at the later layers in the network.