Question 1 Assignment 3

Data Loading and Data Preprocessing:

- **Dataset:**
  CIFAR 10 [1] is the dataset used, It features 10 classes of 60000 colored images of size 32x32. The dataset comes as a training and test set; each set contains 50000 and 10000 images respectively. All the classes in this dataset are mutually exclusive.

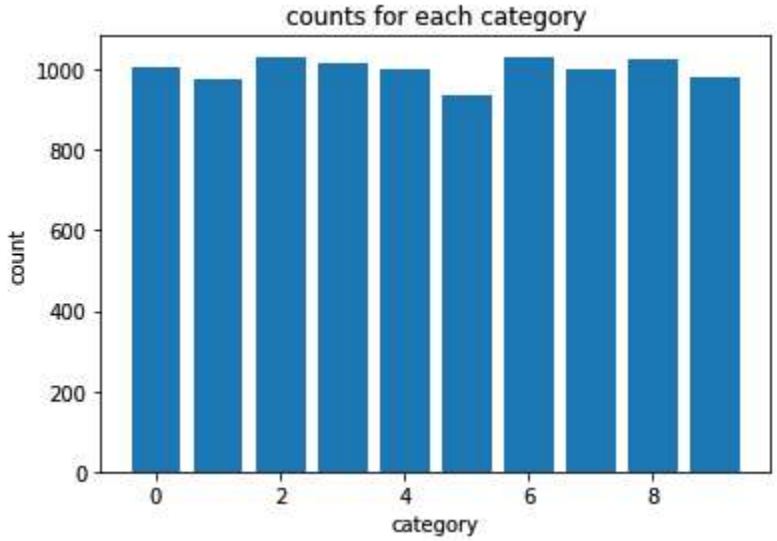
- **Loading the dataset:**
  The dataset is loaded into Google collaboratory notebook using tensorflow's high level keras API. A code snippet for the same looks like this:

```
(x_train, y_train), (x_test, y_test) = tensorflow.keras.datasets.cifar10.load_data()
```

  Shapes of x_train and x_test are (50000,32,32,3) and (10000,32,32,3) respectively. The first shape index represents number of samples, second and third indices represent the image shape and last indices represent channel depth; which is valued as 3 here (explaining RGB image channels).

  Shapes of y_train and y_test are (50000,1) and (10000,1) and for each sample it contains a value from 10 classes {0,1,2,3,4,5,6,7,8,9}.

- **Splitting the dataset:**
  As per the assignment requirement only 20% of the training dataset is used, to select the 20% of train set as training data, data is first shuffled and then 20% of it selected, distribution of each class is also visualized to confirm homogeneity in training data. As can be seen from the plot below 10000 samples are randomly selected from 50000 samples and there are approximately 1000 samples of each category.



  Moreover, the loaded test dataset is split into 50:50 ratio to form new validation and test datasets.Thus the final subsets feature 10000 training samples, 5000 test set samples and 5000 validation samples.

- **Data Preprocessing:**
  - The loaded input data features integer pixel values ranging from 0 to 255, these values are converted into float32 for getting computational advantages.
  - The input dataset features are then normalized from a range of 0 to 255 to 0 to 1, this makes the convergence process of neural networks faster.

- The labels are then converted from integer values {0,1,2,3,4,5,6,7,8,9} to one hot encoded categorical values, thereby making sure the neural network based approach is better able to classify among each class.

Output Layer and Loss Function:

- **Output layer and it's activation function:**
  - The output layer of the network is chosen as a fully connected dense layer. Softmax activation function is used to convert the output of the nodes to represent associated prediction probabilities.
  - The output dense layer features 10 nodes, as the network is supposed to predict probability for each class (10 categories) for a given input. The output of this layer is fed to the softmax function, which gives probabilities for each category between 0 and 1 , which are positively related with the output of the dense layer at the given node.

- **Loss Function:**
  - Categorical cross-entropy is used as a loss function here. Categorical Cross entropy is used since it's a classification problem, where we predict the probabilities of an input image corresponding to each of 10 classes.
  - Cross entropy loss is derived from MLE for classification using logistic regression, categorical cross entropy is in extension of cross entropy (used for binary class classification tasks) for multiple category classification problems. Also the labels are preprocessed as one hot encoded values.

MLP Models

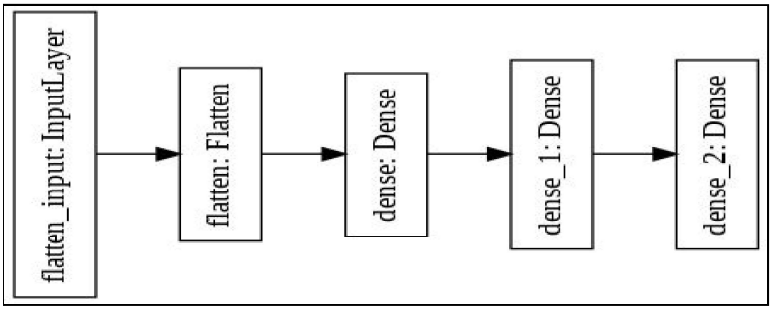
MLP models used for classification:

Various MLP models along with the base model are trained for 5 epochs with Adam optimizer for the given classification problem of CIFAR 10 dataset. The base model with approximately 1.8 million parameters can be described as from the code snippet as :

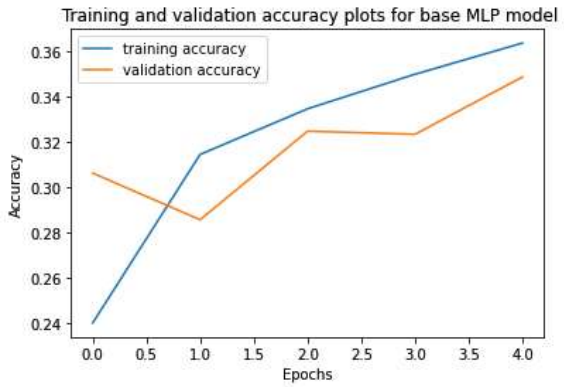
```
model_l3_n512 = tf.keras.models.Sequential([
tf.keras.layers.Flatten(input_shape = (32,32,3)),
tf.keras.layers.Dense(512, activation = 'sigmoid'),
tf.keras.layers.Dense(512, activation = 'sigmoid'),
tf.keras.layers.Dense(10, activation = 'softmax') ])
```

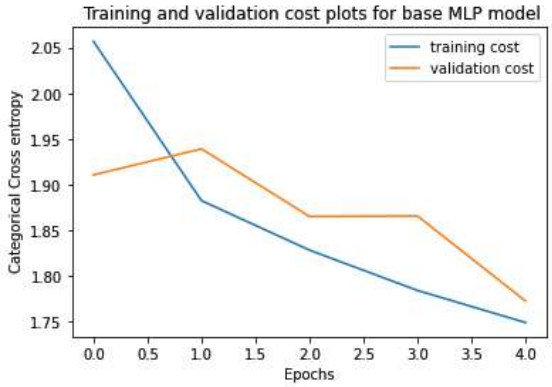
MLP base model architecture and plots:

Base MLP architecture is shown below, it features two Dense(dense, dense_1) layers each having 512 nodes, sigmoid function as activation function followed by output layer (dense_2) which has softmax function as its activation function. The model has approximately 1.8 million parameters. Note that the image data is flattened first, thus 32x32x3 features are converted as 3072x1 features.



MLP base model accuracy and cost plots:





MLP base model analysis:

- Looking at accuracy and cost function plots we can see, model's training and validation accuracy is increasing till 5 epochs. Moreover, the validation and training loss has been decreasing, it can be said that the model is still in the learning phase. No sign of overfitting has been observed.
- Following performance has been achieved by this model on 5 epochs with a batch size of 32:

Training accuracy: 36.38 %

Testing accuracy: 34.88%

Validation accuracy: 34.06%

- Also, looking at above values for training , testing and validation accuracy we can comment that the model has not yet fit the problem well, it looks under-fitted. The gap between training and validation accuracy is not large, also providing evidence of no over-fitting.
- The poor training and validation performance of base MLP model after 5 epochs can be attributed to its inability to utilize spatial information contained in image (as we have to flatten the image first before feeding it to the dense MLP network), moreover model does has any special technique to catch translational invariance presence in images.

MLP model with variation in number of layers and nodes :

Following MLP based models are also trained on the processed dataset, the input layer (32,32,3 features for one sample) is first flattened and is then fed to dense MLP layers followed by output layer featuring 10 nodes:

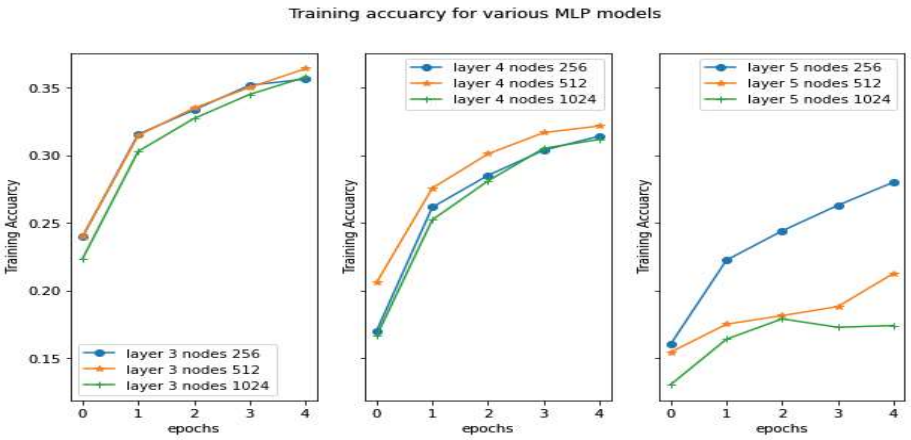
Serial Number	Model Name	Hidden layer nodes, activation	Number of hidden layers	Input layer	Output layer
1	model_l3_n256	Dense, 256, sigmoid	2	32,32,3	10
2	model_l4_n256	Dense, 256, sigmoid	3	32,32,3	10
3	model_l5_n256	Dense, 256, sigmoid	4	32,32,3	10
4	model_l3_n512	Dense, 512, sigmoid	2	32,32,3	10
5	model_l4_n512	Dense, 512, sigmoid	3	32,32,3	10
6	model_l5_n512	Dense, 512, sigmoid	4	32,32,3	10
7	model_l3_n1024	Dense, 1024, sigmoid	2	32,32,3	10
8	model_l4_n1024	Dense, 1024, sigmoid	3	32,32,3	10
9	model_l5_n1024	Dense, 1024, sigmoid	4	32,32,3	10

The performance of these models can be seen from the tables below after 5 epochs can be seen from the table below:

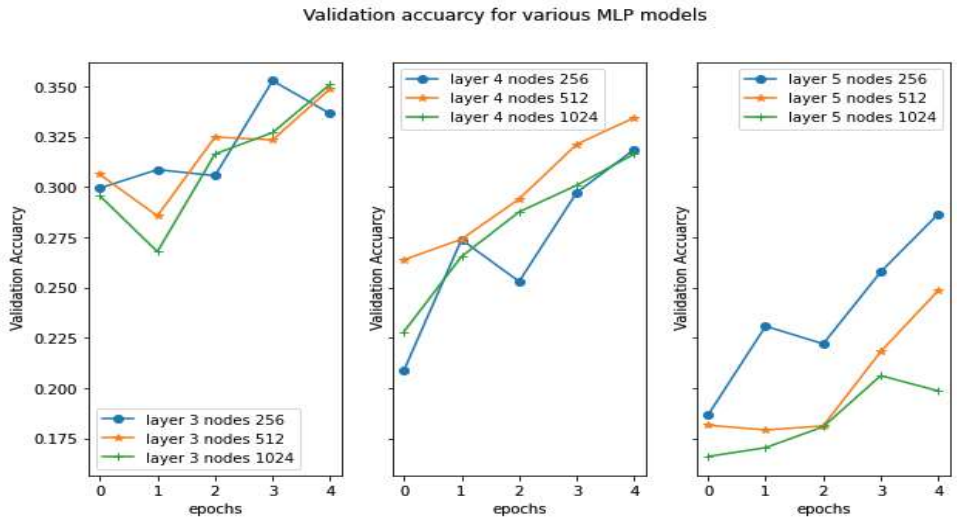
3 Layer model	Hidden layer nodes, activation	Training accuracy	Validation accuracy
model_l3_n256	Dense, 256, sigmoid	35.61%	33.64%
model_l3_n512	Dense, 512, sigmoid	36.38%	34.88%
model_l3_n1024	Dense, 1024, sigmoid	35.76%	35.10%
4 layer model	**Hidden layer nodes, activation**	**Training accuracy**	**Validation accuracy**
model_l4_n256	Dense, 256, sigmoid	31.40%	31.86%
model_l4_n512	Dense, 512, sigmoid	32.15%	33.44%
model_l4_n1024	Dense, 1024, sigmoid	31.16%	31.64%
5 layer model	**Hidden layer nodes, activation**	**Training accuracy**	**Validation accuracy**
model_l5_n256	Dense, 256, sigmoid	28%	28.66%
model_l5_n512	Dense, 512, sigmoid	21.26%	24.86%
model_l5_n1024	Dense, 1024, sigmoid	17.41%	19.86%

Plots for various MLP models with changed number of layer / Neurons:

- **Training accuracy plots:**
  The figure below shows training accuracy for 9 MLP models, here legend represents number of layers in the network and the corresponding number of nodes in the layer.



- **Validation accuracy plots:**
  The figure below shows validation accuracy for 9 MLP models, here legend represents the number of hidden layers in the network and the corresponding number of nodes in the layer.



Observations and comments from various MLP model's performances:

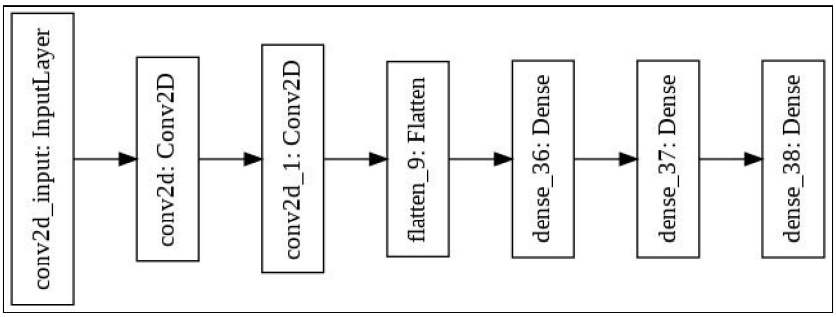
- As more number of layers with the same number of nodes are introduced in the network, the training accuracy of the model decreased and became slow. This can be attributed to the fact that now models have more parameters to train and hence might take more time to train.
- For the same number of layers, most of the time an increment in the number of nodes resulted in slower or almost equal training speed, as can be seen from the above training and validation curves. This further suggests that training speed is negatively related to the number of network parameters (which increase as the number of nodes in a layer increases).
- Training and validation accuracies are very close to each other, hence no overfitting is observed.
- The best training is validation performance is achieved by model with 2 hidden layers and 512 neurons (base model), however models with 2 hidden layers and 256 neurons are supposed to train faster than base model, due to less number of parameters. This might be attributed to the base model's ability to capture more complex and non-linear features compared to less parameter based architecture. However, these values seem pretty close (base model's training and validation accuracy are: 36.38%, 34.33% whereas less parameterized model's training and validation accuracy are: 35.61% and 33.64%). It can be said that selection of the best model would depend on the number of epochs, layers and nodes in the model architecture and is subject to experimentation.

CNN Models

Two CNN models are been used for the analysis purposes:

CNN Model 1:

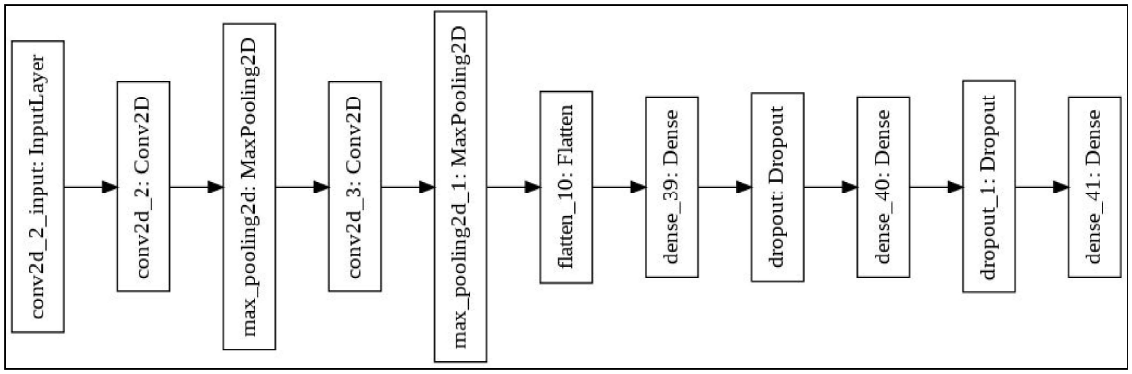
The architecture of the model is shown below, It features two 2D convolutional layers (conv2d, conv2d_1) featuring 64 filters and a window size of 3x3. Both of these layers have rectified linear units as activation functions followed by two fully connected dense layers (dense_36,dense_37) with 512 neurons and sigmoid function as their activation function. The output of the second to last dense layer (dense_38) is fed to the output dense layer featuring 10 nodes and softmax function as its activation function.The model features approximately 25.9 million parameters.



No dropouts/ regularization techniques are used in the architecture. Also The model does not feature pooling layers, making it feature a large number of trainable parameters.

CNN Model2:

The architecture of the model is shown below, It features two 2D convolutional layers (conv2d_d, conv2d_3) featuring 64 filters and a window size of 3x3 each of these convolutional layers is followed by maximum pooling layers (max_pooling2d, max_pooling2d_1). Both of these convolutional layers have rectified linear units as activation functions followed by two fully connected dense layers (dense_39,dense_40) with 512 neurons and sigmoid function as their activation function. The output of the second to last dense layer (dense_41) is fed to the output dense layer featuring 10 nodes and softmax function as its activation function. The model features approximately 1.4 million parameters. CNN model2 features drop out units, and max pooling, which makes this model reduce number of parameters



The performance of CNN model1, CNN model2 and Base MLP after 5 epochs is tabulated below:

Model Name	Training accuracy	Validation accuracy	Test set accuarcy
MLP base model	36.38%	34.88%	34.06%
CNN Model 1	82.80%	52.66%	44.54%
CNN Model 2	56.52%	53.86%	43.86%

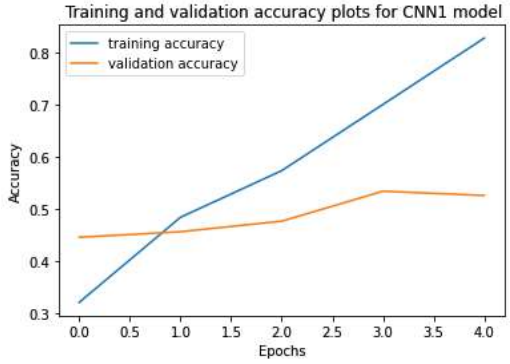
From the above table following comments can be made :

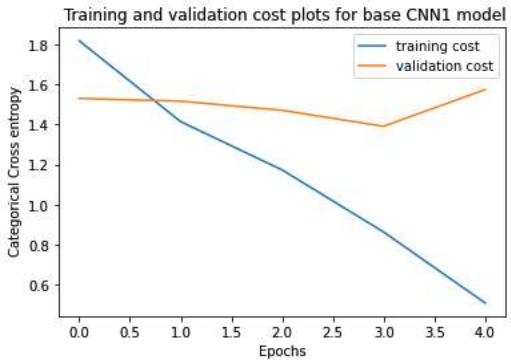
- MLP base model is underfitting with a training accuracy of 36.38% against CNN models training accuracies ; 82.8% and 56.52%. Looks like the MLP model is taking a long time to train, it can be said that it is not able to spatial information from the images. This can be explained as CNN models use inherent properties of images (have shared parameters, and look at a group of pixels together ; based on window size), and hence are better able to extract meaningful features, compared to densely connected MLP networks, as they do not see any order in image pixels.
- CNN model1 looks like it's overfitting, as there is a large gap between training and validation accuracy, more concrete information can be extracted from cost and validation curves for this model. However, the performance of CNN model 1 on training as well as validation set is improved drastically from the MLP base

model.This can be attributed to the application of two 2D convolutional layers before dense layers in the network, the introduction of convolutional layers gained model the ability to extract image features.

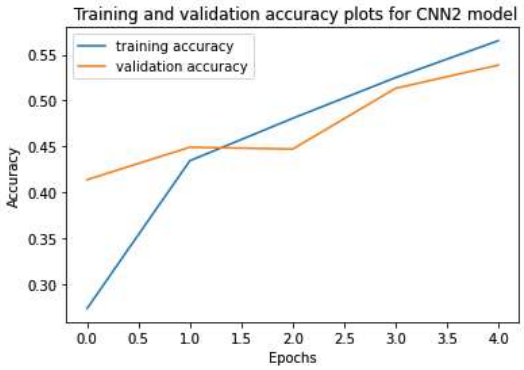
- CNN model1 features approx 25.9 million parameters against 1.8 million parameters of MLP Base model, the large number of parameters to solve this classification might be the cause of overfitting. In addition to this no regularization techniques or parameter reduction techniques such as use of drop outs or pooling layers have been used in the CNN model1, making it computationally expensive to train and prone to overfitting.
- CNN model2 seems robust and looks like it's still learning, the gap between training and validation accuracy is not very large. From the above values of accuracies, it does not look like it's overfitting. This model features approx 1.48 million parameters and is performing better than base MLP model for given 5 epochs.
- It can be said that the introduction of dropouts and max pooling layers in the CNN model2 architecture has reduced the number of parameters in CNN model2 leading to a regularizing effect. On Comparing CNN model2 with base MLP model, it can again be seen that convolutional layers are better able to extract spatial features from the images, thus use of dense layers along with convolutional layers is a good starting point for image classification problems.
- On Comparing performance of CNN model1 and CNN model2, it can be observed that model with more number of parameters (i.e. CNN model 1) in the model makes it prone to overfitting. The introduction of drop out and max pooling layers (also increases the models ability to capture translational invariance in images) in CNN model 2 reduced the number of parameters from approx 25.9 million to 1.48 million parameters, reducing the possibility of the model to overfit.

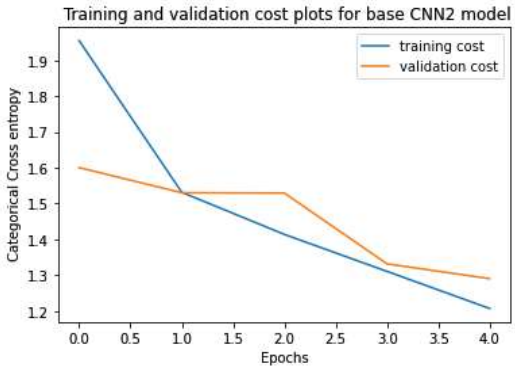
CNN Model 1 Training and validation plots for accuracy and cost:





CNN Model 2 Training and validation plots for accuracy and cost:





On looking at Training and validation plots for 5 epochs for CNN model1 and CNN model2 following observations can be made :

- CNN model 1 is fitting very well on training data, however its performance on validation datas seem comparatively poor, this can be considered as overfitting ; gap between training and validation accuracy as well as training and validation costs looks diverging post 3 epochs, and has kept on increasing post epoch number 2. The reason behind this can be a large number of trainable parameters (approx 25.9 million) which are overfitting the training set.

- CNN model 2 is fitting training and validation well and looks like it's still learning to extract useful features. The gap between training and validation accuracy is not too much. Both training and validation cost values are decreasing with time, representing the model's learning phase. The CNN model2's less number of parameters compared to CNN model1 can be thought as a reason for this behaviour.

Training Time Comparison :

The average training time for 1 epoch for CNN model1 is : approx 3 secs , 9 ms/ step (32 epochs)

The average training time for 1 epoch for CNN model2 is : approx 2 secs , 6 ms/ step (32 epochs)

Note: The time is compared on google collab with a RAM of 1.66 GB (and same for both but unknown GPU configuration).

This large difference in training times for these models can be attributed to the difference in the number of trainable parameters in the model which resulted from use drop outs and max pooling layers in CNN model2 (approx 1.48million parameters) over CNN model1 (approx 25.9 million parameters).

Training the model on more number of epochs :

With the increase in epochs, model starts to earn more complex features.The model tend to optimize cost function on training data at every epoch.This increase in epoch might result in an increase in training accuracy however it might start to overfit the data and loose its ability to generalize on unseen data. I believe training the CNN model1 and CNN model2 for more number of epochs might have following results :

- **CNN model1:** It will continue to overfit, training accuracy will keep on rising and eventually get fixed to some high value, whereas validation accuracy will start oscillating around some value far less than the training accuracy steady state value (probably around 50-60 %).
- **CNN model2:** The fit of CNN model 2 might improve with time, as it seemed still in its learning phase till 5 epochs. Post some number of epochs it might start to overfit or might start to fit even better. One has to look at the accuracy plots for a large number of epochs (the result would be dependent on the number of epochs) to comment on the fit of the CNN model2.

Recommendations to Improve the network

I would like to try following things to improve the network architecture:

- Since the dataset is small, using a model with a large number of parameters might result in overfitting. Therefore I would like to use techniques which help in parameter reduction or provide a regularizing effect. L2/L1 regularization, dropouts, more CNN layers along with pooling layers can be introduced in the network architecture to improve its performance.
- Would like to try some conventional CNN approaches which feature a pattern of increasing the depth and decreasing the width of the receptive field (as were done in Lenet5, Alexnet, VGG-19). These patterns have performed well in the past for some of very famous image datasets and are used in most CNN networks nowadays.
- Instead of flattening the layer, before feeding to desne layers, I would like to try global average pooling/ max-pooling to further reduce the parameters, while keeping some information of extract features in the layer.
- Would also like to try batch normalization technique, as it improves the convergence speed and has a slight regularizing effect. In addition to this data augmentation tricks can also be used to improve the model performance by providing it more data to train,and hence reducing the chances of overfitting.
- Also, I would like to work on tuning the hyperparameters by trying out different learning algorithms like RMS prop, SGD, Adam , along with different learning rate and batch sizes. I would be interested in training the network a bit longer (probably >5 epochs in case of CNN model2) to analyse the learning growth and would like to optimize the number epochs for a better fit.

References:

[1] https://www.cs.toronto.edu/~kriz/cifar.html accessed on July,19,2020.