Shreyas Kapoor | Neural Nets and Deep Learning
Lab Assignment 2

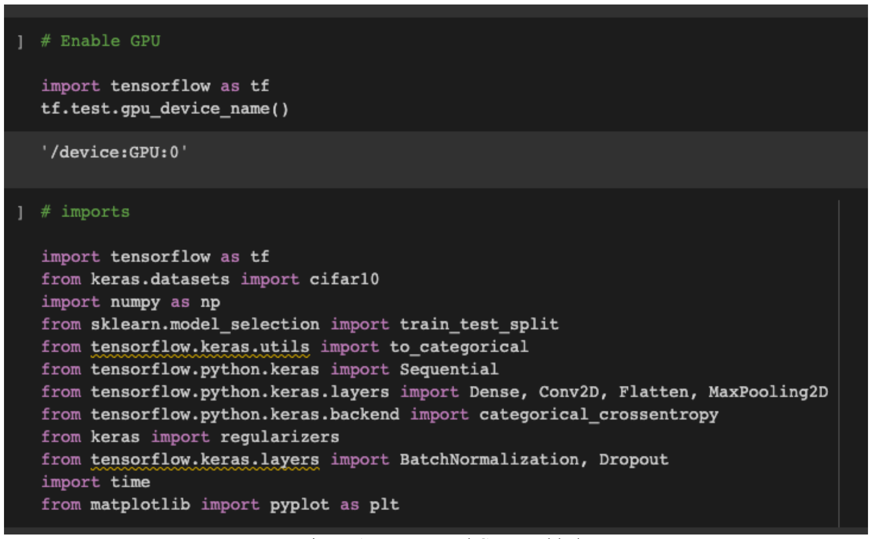
Influence of regularization

- The dataset that was used in this experiment was **ciraf10 dataset** from the **keras** library. The dataset consisted of **50000 images of colored 32*32 images in the training set and 10000 images of same dimension in the test set.** The images are labeled over 10 categories containing labels like airplane, dogs, frog, horse truck etc. The code was developed on colab.research.google.com with GPU enabled.

  The experiments consisted of 9 CNN models where the first 8 models consisted of combinations of different convolutional layers and regularization techniques. The last model was with the best combination of hyperparameters, layers and regularization techniques and was trained over 80% of the data. However, there were some hyperparameters that were consistent across the models. There hyperparameters were

  - **Size of the training data** – 70% of the dataset
  - **Size of the validation set** – 15% of the dataset
  - **No of filters in convolutional layers** – 128
  - **Size of each kernel** – (3,3)
  - **Activation function in convolutional layers** – relu
  - **Activation function in first dense layer** – relu
  - **Activation function in second dense layer** – softmax
  - **Units in first dense layer** – 100
  - **Units in second dense layer** – 10
  - **Optimizer used when compiling the model** – adam
  - **Loss metric in compile layer** – categorical_crossentropy
  - **Batch size while fitting the model** – 32
  - **No of Epochs** – 25

  The first thing that was done was to enable the GPU and testing if that is working. Next, lots of libraries were imported which were useful during model generation.

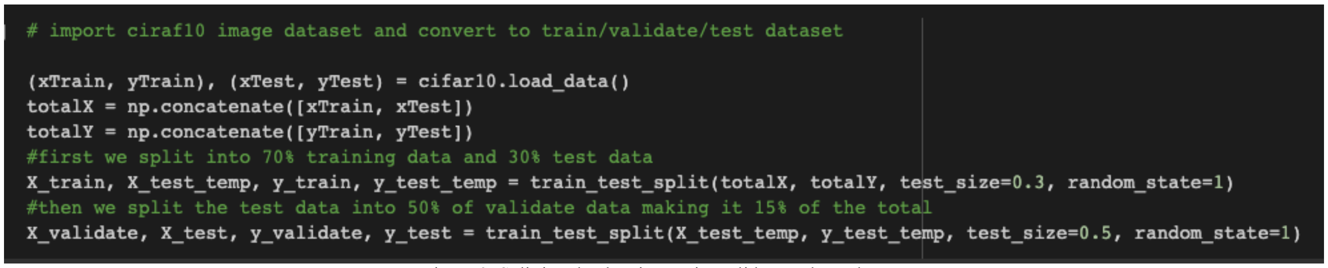


```
# Enable GPU

import tensorflow as tf
tf.test.gpu_device_name()

'/device:GPU:0'
```

```
# imports

import tensorflow as tf
from keras.datasets import cifar10
import numpy as np
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from tensorflow.python.keras import Sequential
from tensorflow.python.keras.layers import Dense, Conv2D, Flatten, MaxPooling2D
from tensorflow.python.keras.backend import categorical_crossentropy
from keras import regularizers
from tensorflow.keras.layers import BatchNormalization, Dropout
import time
from matplotlib import pyplot as plt
```

Figure 1: Imports and GPU enabled

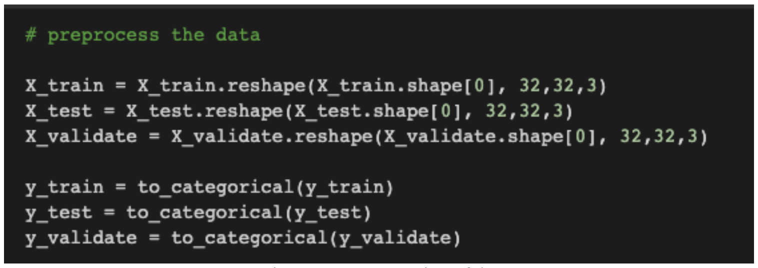
After that, training and testing data was imported. Since the requirement of the model was to train of **70% of the data, validate on 15% of data and finally test on 15% of the data**, I first concatenated the training and testing data using numpy.concatinate() to generate an array of 60000 entries. Next, I split the data into 70% training data and 30% test data. After that, I split the test data into 50% of validate data and test data. This made the validate data to be 15% of the entire dataset(9000 entries).



```
# import ciraf10 image dataset and convert to train/validate/test dataset

(xTrain, yTrain), (xTest, yTest) = cifar10.load_data()
totalX = np.concatenate([xTrain, xTest])
totalY = np.concatenate([yTrain, yTest])
#first we split into 70% training data and 30% test data
X_train, X_test_temp, y_train, y_test_temp = train_test_split(totalX, totalY, test_size=0.3, random_state=1)
#then we split the test data into 50% of validate data making it 15% of the total
X_validate, X_test, y_validate, y_test = train_test_split(X_test_temp, y_test_temp, test_size=0.5, random_state=1)
```

Figure 2: Splitting the data into train, validate and test datasets

After that, I did some preprocessing on the data and the data was ready to be sent to models and train them

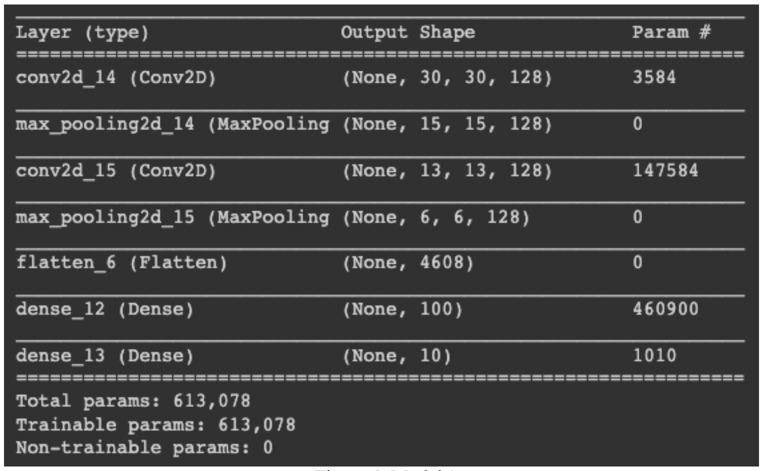


```
# preprocess the data

X_train = X_train.reshape(X_train.shape[0], 32,32,3)
X_test = X_test.reshape(X_test.shape[0], 32,32,3)
X_validate = X_validate.reshape(X_validate.shape[0], 32,32,3)

y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
y_validate = to_categorical(y_validate)
```

Figure 3: Preprocessing of data

Model 1

The first model was created by initializing the Sequential() function from the keras library. Sequential was used because the consecutive layers that are to be added in the model are fed in a sequential way (This logic is same for the rest of the 8 models). The first layer was a Conv2D layer from the keras library. It creates a convolution kernel with the input and produces a tensor of outputs. It had 128 filter of size (3,3), a **relu** activation function and input size of (32, 32, 3). After that, a MaxPooling layer was added which takes the max value present in the widow of size given by user. In our case, the size was (2,2). This combination was repeated once more to create a model with 2 convolutional layers and 2 MaxPooling layers. After this, a Flatten layer was added to flatten the output. After that, a dense layer with 100 units was added which had **relu** activation function. Another dense layer with 10 units and **softmax** activation function was added. This completed the model architecture. The model had 613078 parameters to be trained.

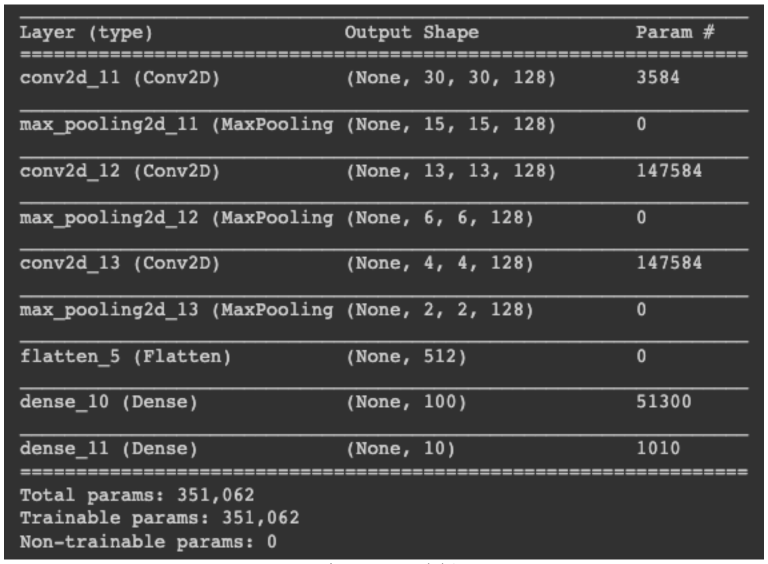


```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_14 (Conv2D)           (None, 30, 30, 128)       3584
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 15, 15, 128)       0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 13, 13, 128)       147584
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 6, 6, 128)         0
_________________________________________________________________
flatten_6 (Flatten)          (None, 4608)              0
_________________________________________________________________
dense_12 (Dense)             (None, 100)               460900
_________________________________________________________________
dense_13 (Dense)             (None, 10)                1010
=================================================================
Total params: 613,078
Trainable params: 613,078
Non-trainable params: 0
```

Figure 4: Model 1

Model 2

The second model was created by initializing the Sequential() function from the keras library. The first layer was a Conv2D layer from the keras library. It had 128 filter of size (3,3), a **relu** activation function and input size of (32, 32, 3). After that, a MaxPooling layer was added with the size (2,2). This combination was repeated twice more to create a model with 3 convolutional layers and 3 MaxPooling layers. After this, a Flatten layer was added to flatten the output. After that, a dense layer with 100 units was added which had **relu** activation function. Another dense layer with 10 units and **softmax** activation function was added. This completed the model architecture. The model had 351062 parameters to be trained.



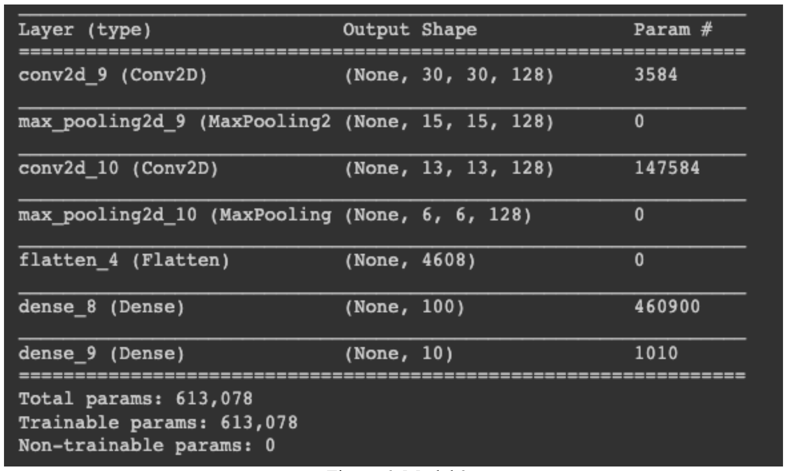
```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_11 (Conv2D)           (None, 30, 30, 128)       3584
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 15, 15, 128)       0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 13, 13, 128)       147584
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 6, 6, 128)         0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 4, 4, 128)         147584
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 2, 2, 128)         0
_________________________________________________________________
flatten_5 (Flatten)          (None, 512)               0
_________________________________________________________________
dense_10 (Dense)             (None, 100)               51300
_________________________________________________________________
dense_11 (Dense)             (None, 10)                1010
=================================================================
Total params: 351,062
Trainable params: 351,062
Non-trainable params: 0
```

Figure 5: Model 2

Model 3

The third model was created by initializing the Sequential() function from the keras library. The first layer was a Conv2D layer from the keras library. It had 128 filter of size (3,3), a **relu** activation function and input size of (32, 32, 3). This layer also had a L2 regularizer with 0.00001 penalty. After that, a MaxPooling layer was added with the size (2,2). This combination was repeated twice more to create a model with 2 convolutional layers and 2 MaxPooling layers. After this, a Flatten layer was added to flatten the output.

After that, a dense layer with 100 units was added which had **relu** activation function. This layer also had a L2 regularizer with 0.00001 penalty. Another dense layer with 10 units and **softmax** activation function was added. This completed the model architecture. The model had 613078 parameters to be trained.

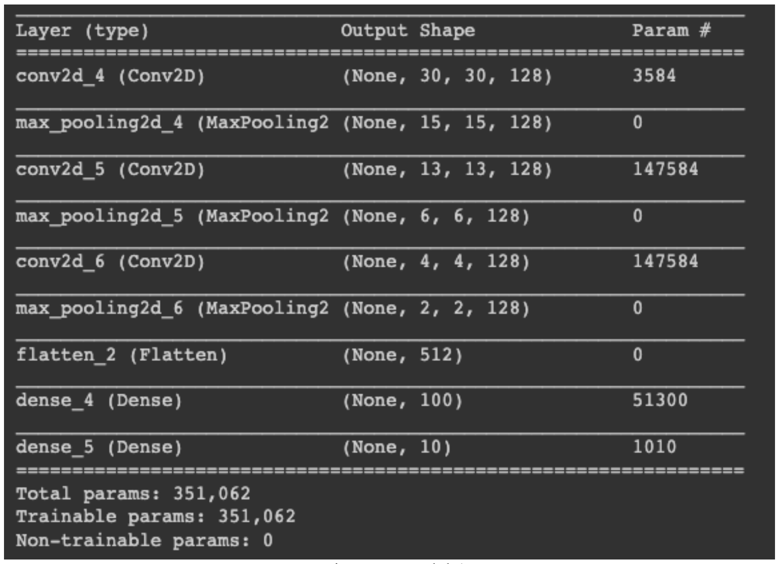


```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_9 (Conv2D)            (None, 30, 30, 128)       3584
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 15, 15, 128)       0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 13, 13, 128)       147584
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 6, 6, 128)         0
_________________________________________________________________
flatten_4 (Flatten)          (None, 4608)              0
_________________________________________________________________
dense_8 (Dense)              (None, 100)               460900
_________________________________________________________________
dense_9 (Dense)              (None, 10)                1010
=================================================================
Total params: 613,078
Trainable params: 613,078
Non-trainable params: 0
```

Figure 6: Model 3

Model 4

The fourth model was created by initializing the Sequential() function from the keras library. The first layer was a Conv2D layer from the keras library. It had 128 filter of size (3,3), a **relu** activation function and input size of (32, 32, 3). This layer also had a L2 regularizer with 0.00001 penalty. After that, a MaxPooling layer was added with the size (2,2). This combination was repeated twice more to create a model with 3 convolutional layers and 3 MaxPooling layers. After this, a Flatten layer was added to flatten the output. After that, a dense layer with 100 units was added which had **relu** activation function. This layer also had a L2 regularizer with 0.00001 penalty. Another dense layer with 10 units and **softmax** activation function was added. This completed the model architecture. The model had 351062 parameters to be trained.

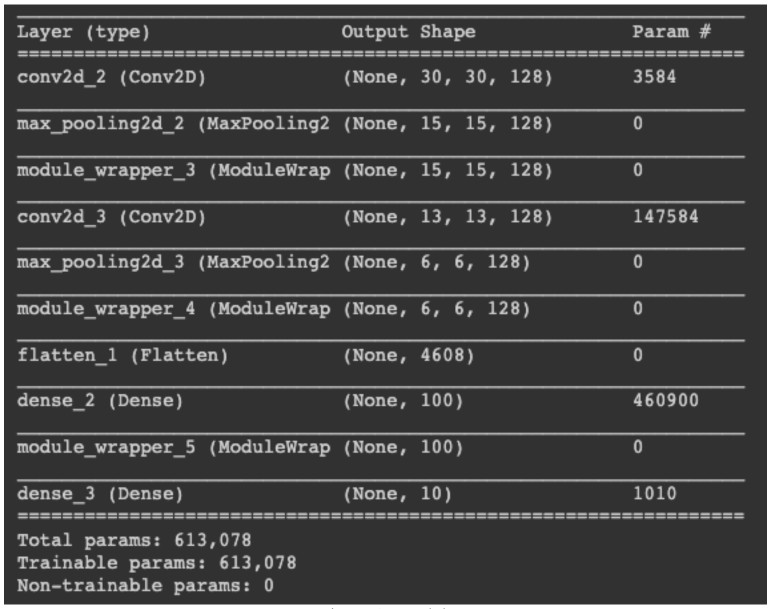


```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 30, 30, 128)       3584
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 15, 15, 128)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 13, 13, 128)       147584
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 6, 6, 128)         0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 4, 4, 128)         147584
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 2, 2, 128)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 512)               0
_________________________________________________________________
dense_4 (Dense)              (None, 100)               51300
_________________________________________________________________
dense_5 (Dense)              (None, 10)                1010
=================================================================
Total params: 351,062
Trainable params: 351,062
Non-trainable params: 0
```

Figure 7: Model 4

Model 5

The fifth model was created by initializing the Sequential() function from the keras library. The first layer was a Conv2D layer from the keras library. It had 128 filter of size (3,3), a **relu** activation function and input size of (32, 32, 3). After that, a MaxPooling layer was added with the size (2,2). After that, a dropout layer was added with rate of 0.2. This combination was repeated twice more to create a model with 2 convolutional layers and 2 MaxPooling layers and 2 dropout layers. After this, a Flatten layer was added to flatten the output. After that, a dense layer with 100 units was added which had **relu** activation function. After that, a dropout layer was added with rate of 0.2. Another dense layer with 10 units and **softmax** activation function was added. This completed the model architecture. The model had 613078 parameters to be trained.

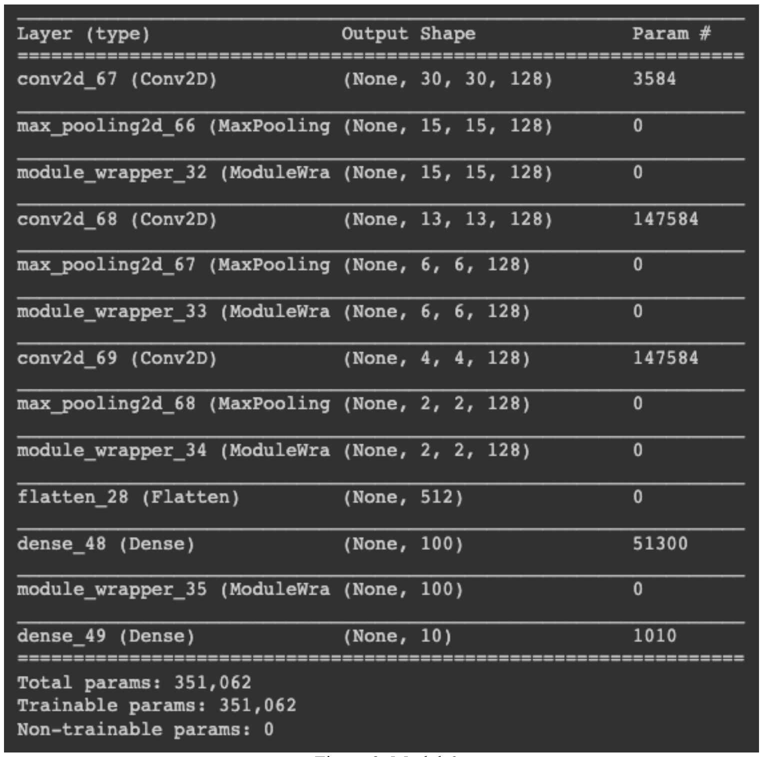


```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 30, 30, 128)       3584
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 15, 15, 128)       0
_________________________________________________________________
module_wrapper_3 (ModuleWrap (None, 15, 15, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 13, 13, 128)       147584
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 6, 6, 128)         0
_________________________________________________________________
module_wrapper_4 (ModuleWrap (None, 6, 6, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 4608)              0
_________________________________________________________________
dense_2 (Dense)              (None, 100)               460900
_________________________________________________________________
module_wrapper_5 (ModuleWrap (None, 100)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1010
=================================================================
Total params: 613,078
Trainable params: 613,078
Non-trainable params: 0
```

Figure 8: Model 5

Model 6

The sixth model was created by initializing the Sequential() function from the keras library. The first layer was a Conv2D layer from the keras library. It had 128 filter of size (3,3), a **relu** activation function and input size of (32, 32, 3). After that, a MaxPooling layer was added with the size (2,2). After that, a dropout layer was added with rate of 0.2. This combination was repeated twice more to create a model with 3 convolutional layers and 3 MaxPooling layers and 3 dropout layers. After this, a Flatten layer was added to flatten the output. After that, a dense layer with 100 units was added which had **relu** activation function. After that, a dropout layer was added with rate of 0.2. Another dense layer with 10 units and **softmax** activation function was added. This completed the model architecture. The model had 351062 parameters to be trained.

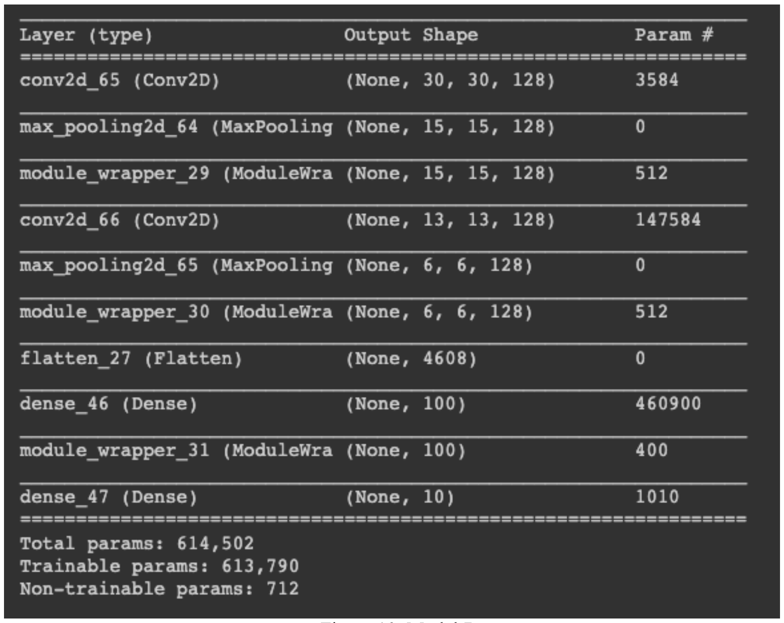


```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_67 (Conv2D)           (None, 30, 30, 128)       3584      
_________________________________________________________________
max_pooling2d_66 (MaxPooling (None, 15, 15, 128)       0         
_________________________________________________________________
module_wrapper_32 (ModuleWra (None, 15, 15, 128)       0         
_________________________________________________________________
conv2d_68 (Conv2D)           (None, 13, 13, 128)       147584    
_________________________________________________________________
max_pooling2d_67 (MaxPooling (None, 6, 6, 128)         0         
_________________________________________________________________
module_wrapper_33 (ModuleWra (None, 6, 6, 128)         0         
_________________________________________________________________
conv2d_69 (Conv2D)           (None, 4, 4, 128)         147584    
_________________________________________________________________
max_pooling2d_68 (MaxPooling (None, 2, 2, 128)         0         
_________________________________________________________________
module_wrapper_34 (ModuleWra (None, 2, 2, 128)         0         
_________________________________________________________________
flatten_28 (Flatten)         (None, 512)               0         
_________________________________________________________________
dense_48 (Dense)             (None, 100)               51300     
_________________________________________________________________
module_wrapper_35 (ModuleWra (None, 100)               0         
_________________________________________________________________
dense_49 (Dense)             (None, 10)                1010      
=================================================================
Total params: 351,062
Trainable params: 351,062
Non-trainable params: 0
```

Figure 9: Model 6

Model 7

The seventh model was created by initializing the Sequential() function from the keras library. The first layer was a Conv2D layer from the keras library. It had 128 filter of size (3,3), a **relu** activation function and input size of (32, 32, 3). After that, a MaxPooling layer was added with the size (2,2). After that, a Batch Normalization layer was added. This combination was repeated twice more to create a model with 2 convolutional layers and 2 MaxPooling layers and 2 Batch Normalization layers. After this, a Flatten layer was added to flatten the output. After that, a dense layer with 100 units was added which had **relu** activation function. After that, a Batch Normalization layer was added. Another dense layer with 10 units and **softmax** activation function was added. This completed the model architecture. The model had 614502 parameters out of which, 712 were untrainable parameters.

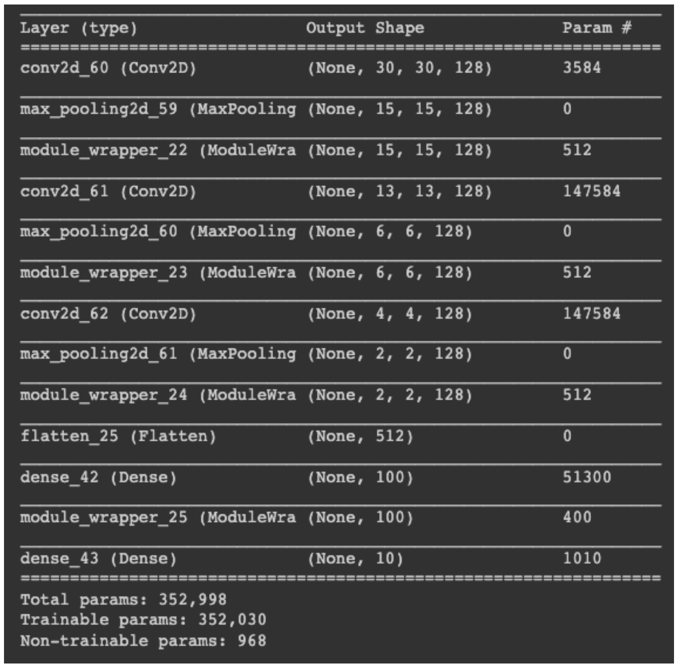


```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_65 (Conv2D)           (None, 30, 30, 128)       3584
_________________________________________________________________
max_pooling2d_64 (MaxPooling (None, 15, 15, 128)       0
_________________________________________________________________
module_wrapper_29 (ModuleWra (None, 15, 15, 128)       512
_________________________________________________________________
conv2d_66 (Conv2D)           (None, 13, 13, 128)       147584
_________________________________________________________________
max_pooling2d_65 (MaxPooling (None, 6, 6, 128)         0
_________________________________________________________________
module_wrapper_30 (ModuleWra (None, 6, 6, 128)         512
_________________________________________________________________
flatten_27 (Flatten)         (None, 4608)              0
_________________________________________________________________
dense_46 (Dense)             (None, 100)               460900
_________________________________________________________________
module_wrapper_31 (ModuleWra (None, 100)               400
_________________________________________________________________
dense_47 (Dense)             (None, 10)                1010
=================================================================
Total params: 614,502
Trainable params: 613,790
Non-trainable params: 712
```

Figure 10: Model 7

Model 8

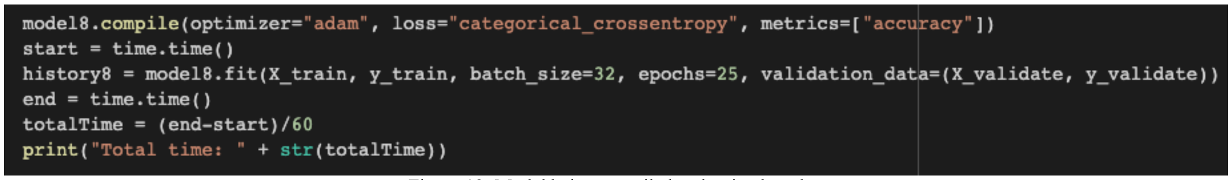
The eighth model was created by initializing the Sequential() function from the keras library. The first layer was a Conv2D layer from the keras library. It had 128 filter of size (3,3), a **relu** activation function and input size of (32, 32, 3). After that, a MaxPooling layer was added with the size (2,2). After that, a Batch Normalization layer was added. This combination was repeated twice more to create a model with 3 convolutional layers and 3 MaxPooling layers and 3 Bath Normalization layers. After this, a Flatten layer was added to flatten the output. After that, a dense layer with 100 units was added which had **relu** activation function. After that, a Batch Normalization layer was added. Another dense layer with 10 units and **softmax** activation function was added. This completed the model architecture. The model had 352998 parameters out of which, 968 were untrainable parameters.



```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_60 (Conv2D)           (None, 30, 30, 128)       3584
_________________________________________________________________
max_pooling2d_59 (MaxPooling (None, 15, 15, 128)       0
_________________________________________________________________
module_wrapper_22 (ModuleWra (None, 15, 15, 128)       512
_________________________________________________________________
conv2d_61 (Conv2D)           (None, 13, 13, 128)       147584
_________________________________________________________________
max_pooling2d_60 (MaxPooling (None, 6, 6, 128)         0
_________________________________________________________________
module_wrapper_23 (ModuleWra (None, 6, 6, 128)         512
_________________________________________________________________
conv2d_62 (Conv2D)           (None, 4, 4, 128)         147584
_________________________________________________________________
max_pooling2d_61 (MaxPooling (None, 2, 2, 128)         0
_________________________________________________________________
module_wrapper_24 (ModuleWra (None, 2, 2, 128)         512
_________________________________________________________________
flatten_25 (Flatten)         (None, 512)               0
_________________________________________________________________
dense_42 (Dense)             (None, 100)               51300
_________________________________________________________________
module_wrapper_25 (ModuleWra (None, 100)               400
_________________________________________________________________
dense_43 (Dense)             (None, 10)                1010
=================================================================
Total params: 352,998
Trainable params: 352,030
Non-trainable params: 968
```

Figure 11: Model 8

After the models was build, each model was compiled using the adam optimizer, loss as categorical_crossentropy and metrics used was accuracy. Then the model was trained on the training data set with batch size as 32 and number of epochs as 25. Validation dataset was also sent to the model to validate the accuracy of the model parameters. Time was also calculated for how long it took for the model to get trained. Accuracy plots were also generated which will be discussed later.



```
model8.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
start = time.time()
history8 = model8.fit(X_train, y_train, batch_size=32, epochs=25, validation_data=(X_validate, y_validate))
end = time.time()
totalTime = (end-start)/60
print("Total time: " + str(totalTime))
```

Figure 12: Model being compiled and trained on data.

Choosing from the best accuracy on the model, model 8 was chosen because it had the highest accuracy on validation data. A new model 9 was created and finally the test data was evaluated on that.

- The training time in minutes for each model was as follows

Table 1: Model and their training time (in mins)

Model 1	3.38
Model 2	3.38
Model 3	2.93
Model 4	3.30
Model 5	3.09
Model 6	3.36
Model 7	3.38
Model 8	4.38

The following are the graphs of training data and validate data on each model. X axis contains the number of epochs and y axis contains the accuracy of the dataset in that model.

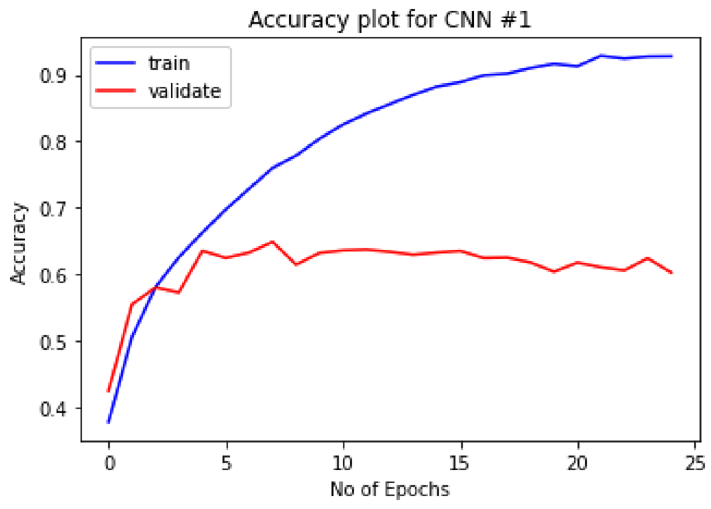


Figure 13: Accuracy vs No of Epochs graph of Model 1

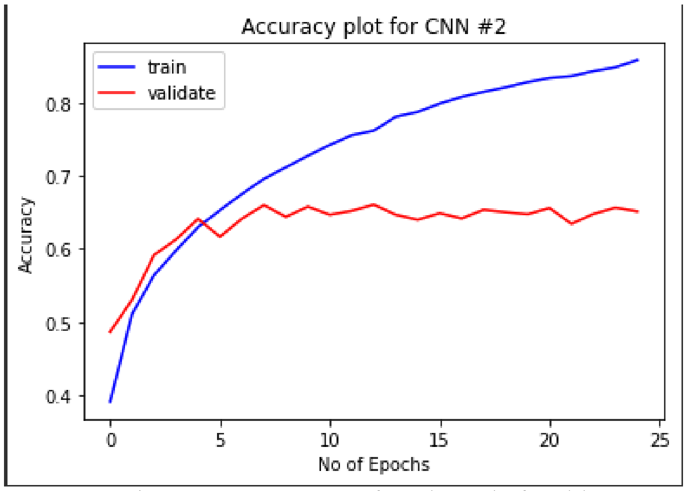


Figure 14: Accuracy vs No of Epochs graph of Model 2

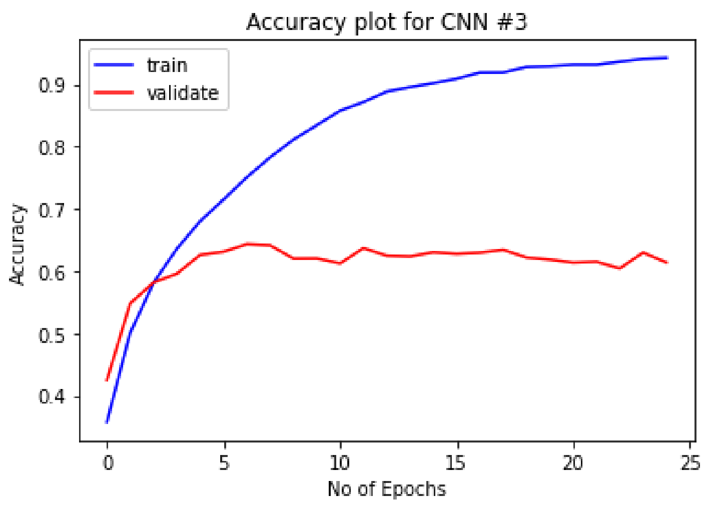


Figure 53: Accuracy vs No of Epochs graph of Model 13

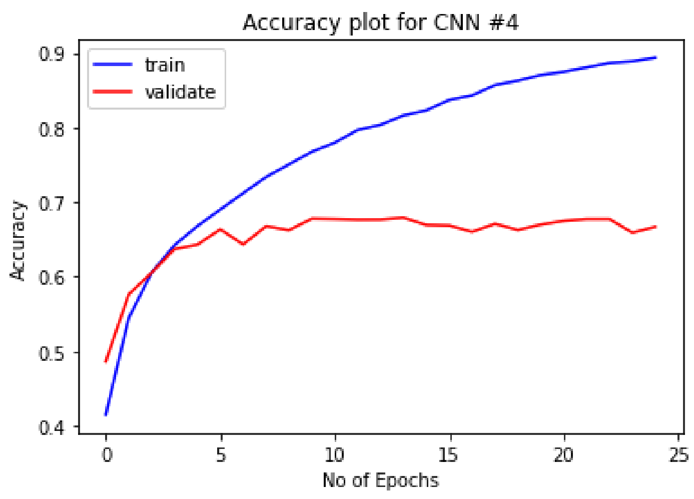


Figure 16: Accuracy vs No of Epochs graph of Model 4

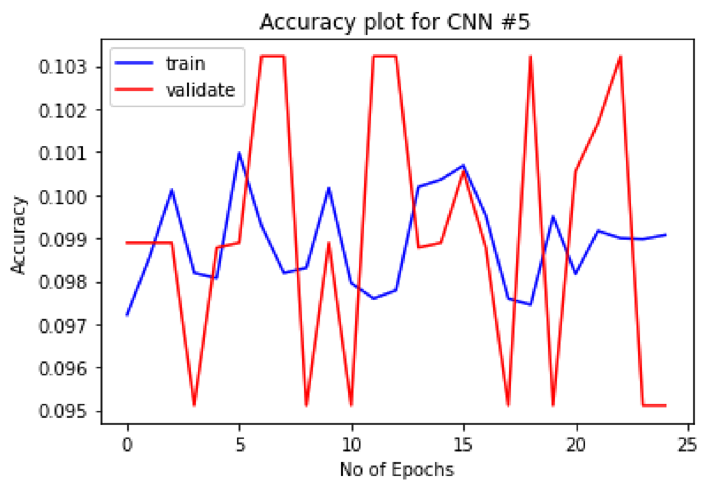


Figure 17: Accuracy vs No of Epochs graph of Model 15

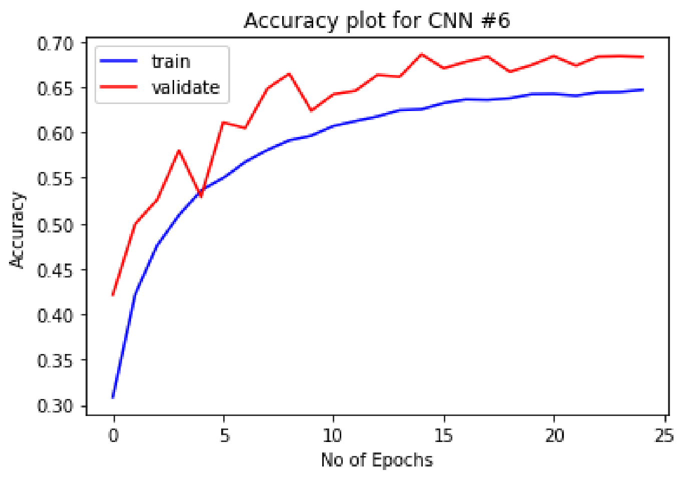


Figure 18: Accuracy vs No of Epochs graph of Model 6

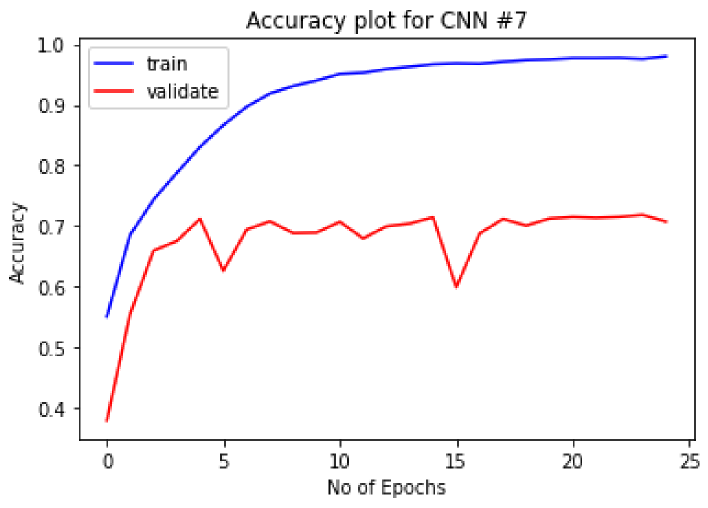


Figure 19: Accuracy vs No of Epochs graph of Model 17

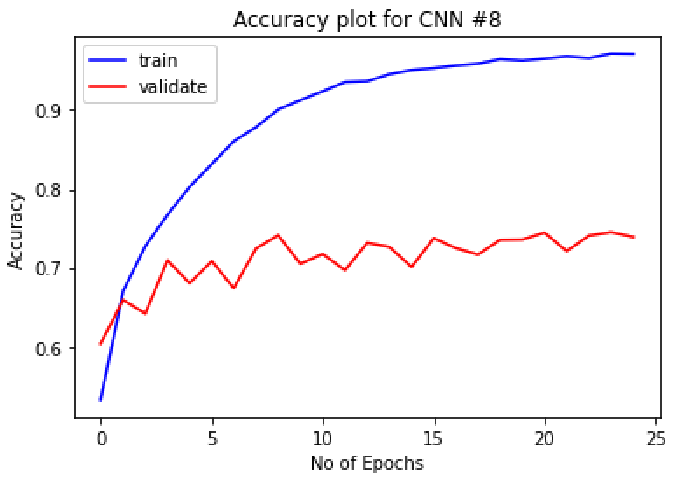


Figure 20: Accuracy vs No of Epochs graph of Model 8

- The following table depicts the accuracy of the models on the training data and validation data

Table 2 Comparison of Training data accuracy and validation data accuracy on models

Model	Train Accuracy	Validation Accuracy
Model 1	0.92	0.60
Model 2	0.85	0.65
Model 3	0.94	0.61
Model 4	0.89	0.67
Model 5	0.09	0.09
Model 6	0.64	0.68
Model 7	0.97	0.70
Model 8	0.97	0.73

From the table it is clear that model 8 worked best on the validation dataset. This model was then selected as the final model and the model was trained on 85% of the dataset. The test data consisted of 15% of the dataset. **This model finally gave an accuracy of 0.73.**

- Some of the trends that I noticed are discussed below
  - **No of Convolutional Layers and Regularization Techniques:** From the accuracy table, we can see that in general, models with 3 convolution layers and 3 max pooling layers worked better than two convolution layers and 2 max pooling layers. This can be attributed to the fact that the more time we give to the data to get trained on, the model will learn more things and hence will work better. Another notable thing was that with the models that used regularization, the validation accuracy was higher than the models without regularization. This was also expected as regularization introduces penalties which make the models learn better by penalizing it in case it's going to fit incorrectly.

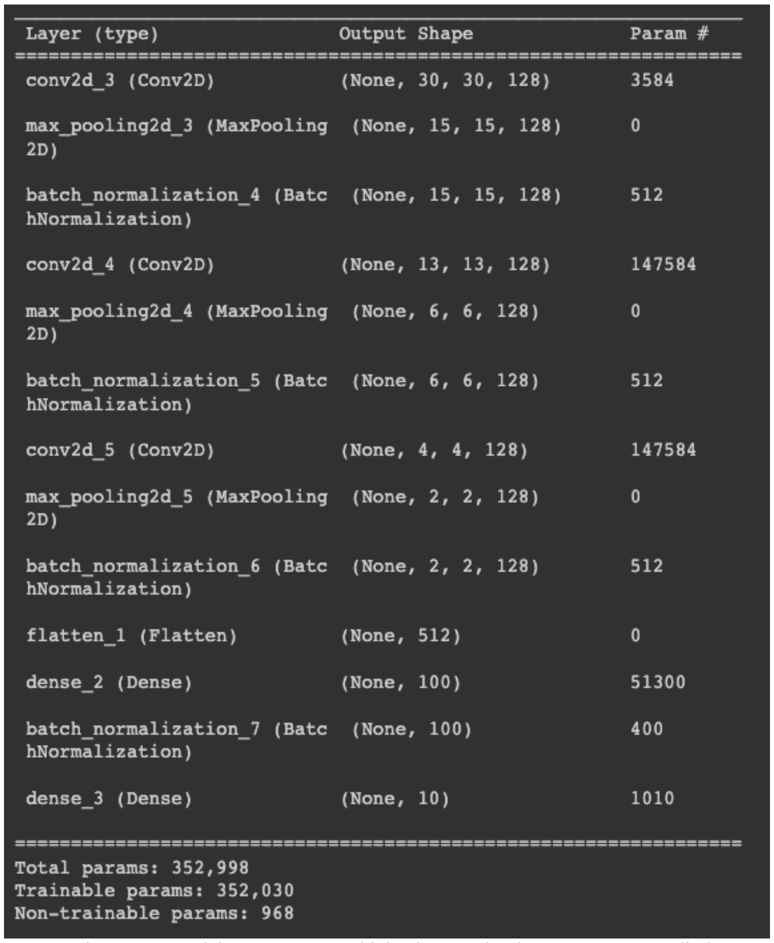
  - **No of parameters:** I believed that if we increase the number of layers, the number of parameters that needed to be trained increases. That belief was shattered in this experiment. Models 2,4,6,8 which had 3 convolution layers, and 3 maxPooling layers had a smaller number of parameters than models 1,3,5,7 which had 2 convolution layers, and 2 maxPooling layers. I forgot to realize that MaxPooling layer takes the max value from the pool of values present in the size of the pooling window. This will obviously reduce the size of the input value for next layer. So, before we send the input to the flatten layer, increasing the convolution layer and max pooling layer reduces the input size by 4 times (because I used 2*2 pooling size).

  - **Training time –** Apart from the models without regularization, where there was no effect of increasing the convolution Layers and maxPooling Layers on training time, other models showed a general trend that as we increase the no of layers, training time increase. This happens because as we increase the no of layers, the data is being passed from more layers and new computations are taking pace. This takes extra time and hence the extra time in training the models as well.

Interpreting CNN Representations

- To work on this experiment, I first imported all the necessary imports. After that, I downloaded the cifar10 dataset and then concatenated it to make a total sample size of 60,000 images. The next step was to use train_test_split to split into 15% test data and 85% train data. The next step was to preprocess the data by reshaping it and changing into categorical numpy array. After that, the model was created.

  The model was created by initializing the Sequential() function from the keras library. The first layer was a Conv2D layer from the keras library. It had 128 filter of size (3,3), a **relu** activation function and input size of (32, 32, 3). After that, a MaxPooling layer was added with the size (2,2). After that, a Batch Normalization layer was added. This combination was repeated twice more to create a model with 3 convolutional layers and 3 MaxPooling layers and 3 Bath Normalization layers. After this, a Flatten layer was added to flatten the output. After that, a dense layer with 100 units was added which had **relu** activation function. After that, a Batch Normalization layer was added. Another dense layer with 10 units and **softmax** activation function was added. This completed the model architecture. The model had 352998 parameters out of which, 968 were untrainable parameters.



```
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 30, 30, 128)       3584

 max_pooling2d_3 (MaxPooling  (None, 15, 15, 128)      0
 2D)

 batch_normalization_4 (Batc  (None, 15, 15, 128)      512
 hNormalization)

 conv2d_4 (Conv2D)           (None, 13, 13, 128)       147584

 max_pooling2d_4 (MaxPooling  (None, 6, 6, 128)        0
 2D)

 batch_normalization_5 (Batc  (None, 6, 6, 128)        512
 hNormalization)

 conv2d_5 (Conv2D)           (None, 4, 4, 128)         147584

 max_pooling2d_5 (MaxPooling  (None, 2, 2, 128)        0
 2D)

 batch_normalization_6 (Batc  (None, 2, 2, 128)        512
 hNormalization)

 flatten_1 (Flatten)         (None, 512)               0

 dense_2 (Dense)             (None, 100)               51300

 batch_normalization_7 (Batc  (None, 100)              400
 hNormalization)

 dense_3 (Dense)             (None, 10)                1010

=================================================================
Total params: 352,998
Trainable params: 352,030
Non-trainable params: 968
```

Figure 22: Model Summary on which Class Activation Maps were studied

  After the model was trained, I took an image of car

Figure 23: Image of the car

And did preprocessing on it convert into an array. The original size of the image was 480*293. Since our model required a 32*32 input, resizing it drastically compromised it’s size.

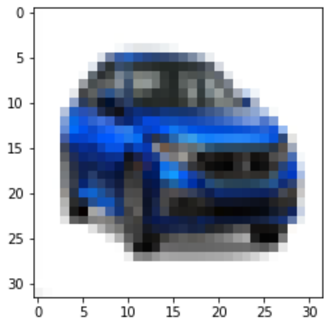
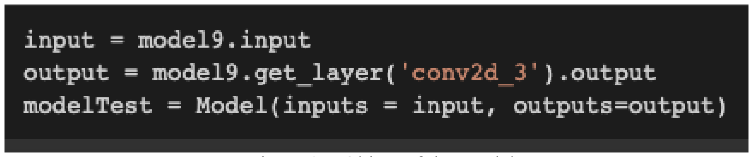


Figure 24: Image of the car after resizing it into 32*32 image

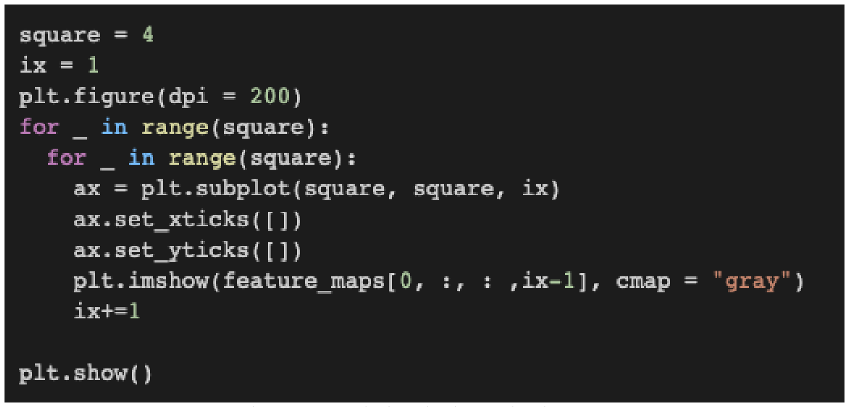
Next, I created an object of the model where input was same as the input for original model and output as the ouput given by first conv2d layer.



```
input = model9.input
output = model9.get_layer('conv2d_3').output
modelTest = Model(inputs = input, outputs=output)
```

Figure 25: Object of the Model

After that, I called the predict function to predict the class of the image. After that, subplots of the activation maps were plotted.



```
square = 4
ix = 1
plt.figure(dpi = 200)
for _ in range(square):
  for _ in range(square):
    ax = plt.subplot(square, square, ix)
    ax.set_xticks([])
    ax.set_yticks([])
    plt.imshow(feature_maps[0, :, : ,ix-1], cmap = "gray")
    ix+=1

plt.show()
```

Figure 26: Code for plotting activation maps

Since the experiment required us to do a comparative analysis on two layers, the last part of creating an object of model and generating the subplots was repeated fort another layer.

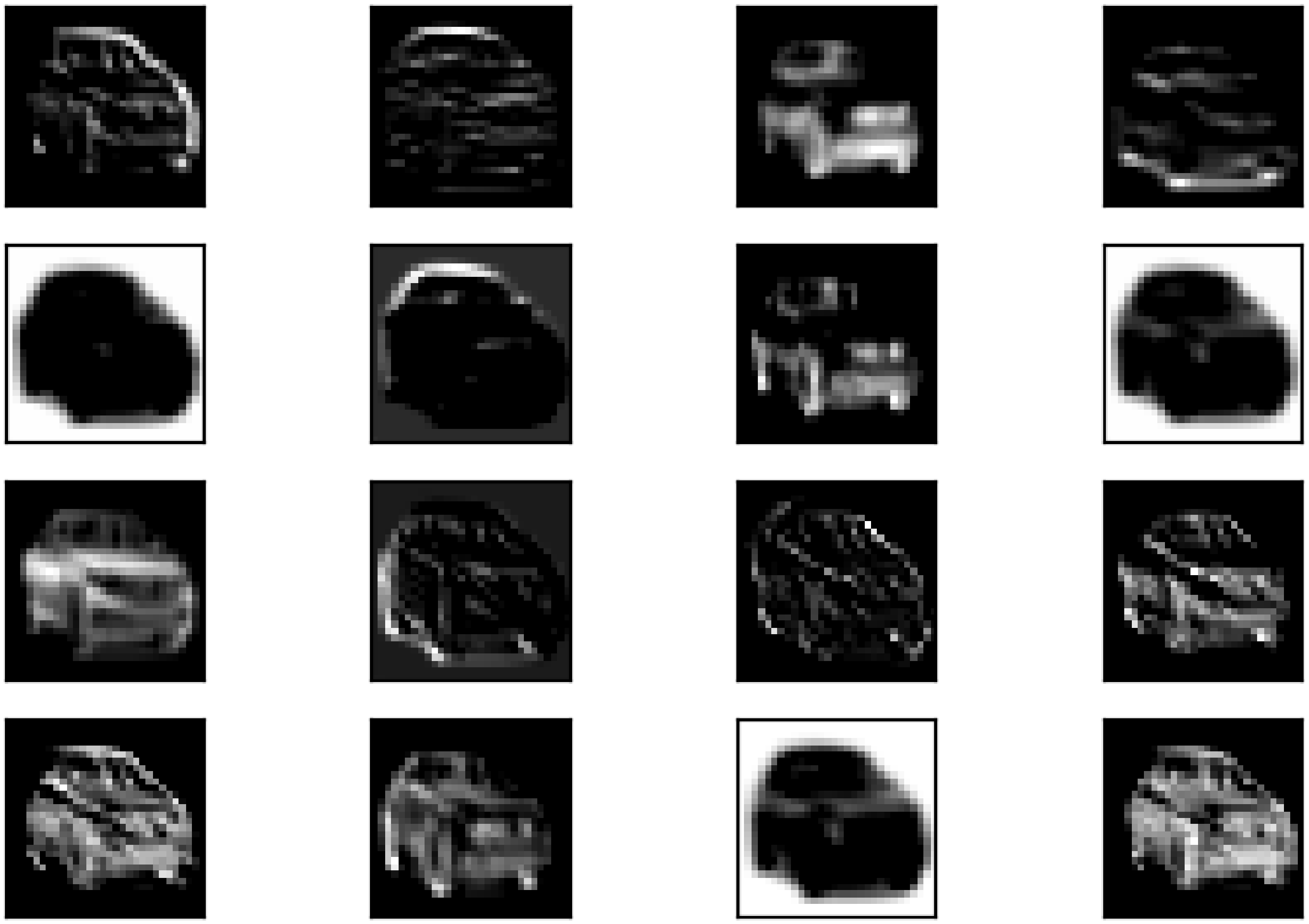


Figure 27: Activation Maps from output of first convolution layer

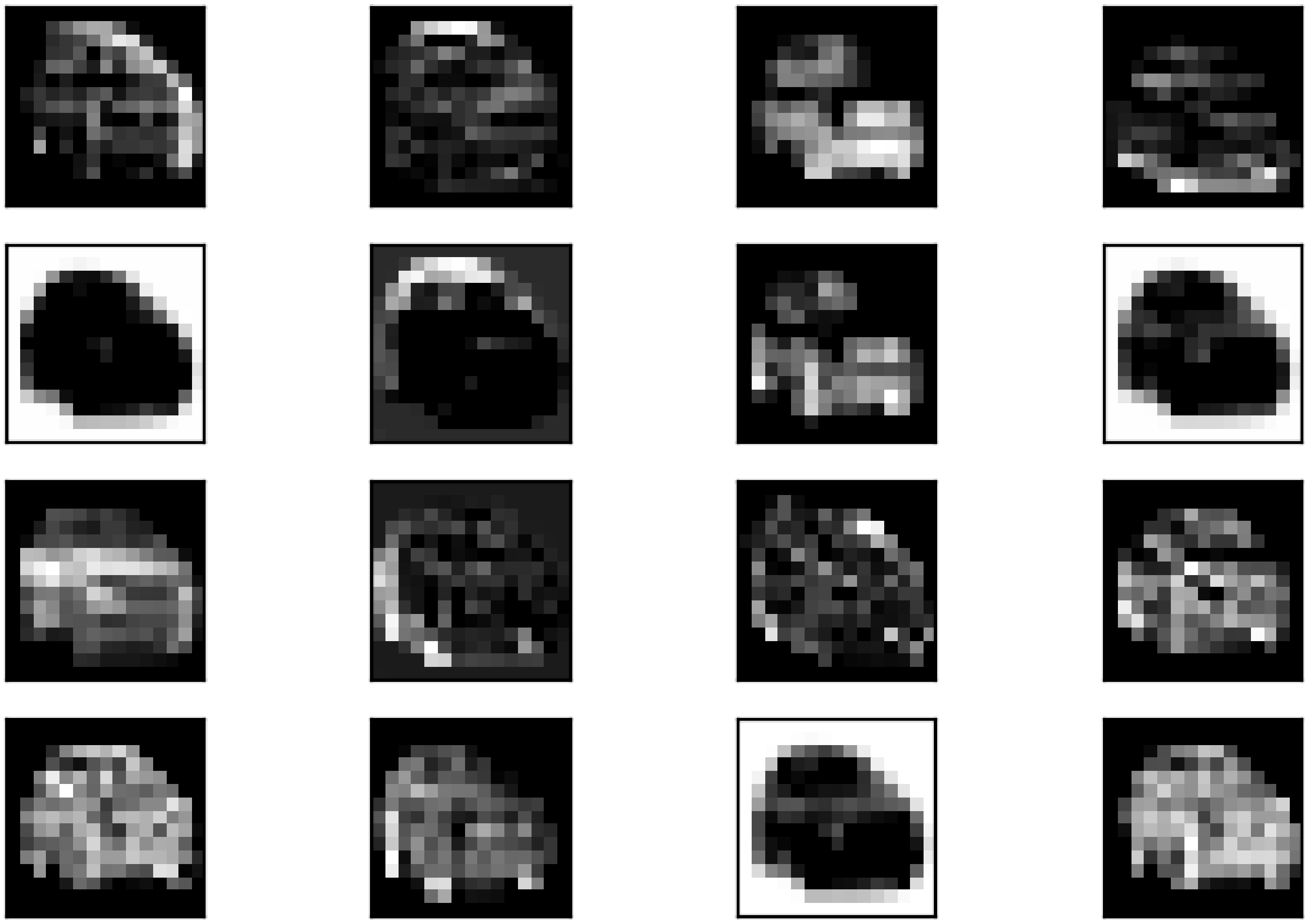


Figure 28: Activation Maps from output of first batch Normalization Layer

Analysis of the trends

- **How does what is learned at different layers of the network differs**

  From the above results, what I realized is happening is what features of the image are important to be carried forward by the model to predict what the image is about. From the top left image in the first plot, we find that the model is trying to figure out the end of the car from upper right corner. Since the boundary of the car is important, this detail is carried forward and in the second plot, we can see that the model is trying to blur out the rest of the details as they are not important.

- **What did the model learn**

  From the above subplots, what I could infer is that model is trying to learn, layer by layer, what features of the image are important and what are not. This is being done to predict what the image is about. For example, in the Figure 27, we can see that after the first layer, the model is figuring out, with the help of different filters, what features are there in the image of car, like types, outline of the car headlights etc. In the second figure, we can see that although more blurred, the model is trying to highlight the more prominent features which will help it classify it as car.

```python
# imports

import tensorflow as tf
from keras.datasets import cifar10
import numpy as np
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from tensorflow.python.keras import Sequential
from tensorflow.python.keras.layers import Dense, Conv2D, Flatten, MaxPooling2D
from tensorflow.python.keras.backend import categorical_crossentropy
from keras import regularizers
from tensorflow.keras.layers import BatchNormalization, Dropout
import time
from matplotlib import pyplot as plt

# import ciraf10 image dataset and convert to train/validate/test dataset

(xTrain, yTrain), (xTest, yTest) = cifar10.load_data()
totalX = np.concatenate([xTrain, xTest])
totalY = np.concatenate([yTrain, yTest])
#first we split into 70% training data and 30% test data
X_train, X_test_temp, y_train, y_test_temp = train_test_split(totalX, totalY, test_size=0.3, random_state=1)
#then we split the test data into 50% of validate data making it 15% of the total
X_validate, X_test, y_validate, y_test = train_test_split(X_test_temp, y_test_temp, test_size=0.5, random_state=1)

# preprocess the data

X_train = X_train.reshape(X_train.shape[0], 32,32,3)
X_test = X_test.reshape(X_test.shape[0], 32,32,3)
X_validate = X_validate.reshape(X_validate.shape[0], 32,32,3)

y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
y_validate = to_categorical(y_validate)

#create CNN 1 with 2 conv2d layers and no regularization

model1 = Sequential()

model1.add(Conv2D(128, (3,3), activation='relu',input_shape=(32,32,3)))
model1.add(MaxPooling2D(pool_size=(2,2)))
model1.add(Conv2D(128, (3,3), activation='relu'))
model1.add(MaxPooling2D(pool_size=(2,2)))
model1.add(Flatten())
model1.add(Dense(100, activation = "relu"))
model1.add(Dense(10, activation = "softmax"))
model1.summary()

model1.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
start = time.time()
history1 = model1.fit(X_train, y_train, batch_size=32, epochs=25, validation_data=(X_validate, y_validate))
end = time.time()
totalTime = (end-start)/60
print("Total time: " + str(totalTime))

plt.plot(history1.history['accuracy'], color="blue", label="train")
plt.plot(history1.history['val_accuracy'], color="red", label="validate")
plt.xlabel("No of Epochs")
plt.ylabel("Accuracy")
plt.title("Accuracy plot for CNN #1")
plt.legend()
plt.show()

#create CNN 2 with 3 conv2d layers and no regularization

model2 = Sequential()

model2.add(Conv2D(128, (3,3), activation='relu', input_shape=(32,32,3)))
model2.add(MaxPooling2D(pool_size=(2,2)))
model2.add(Conv2D(128, (3,3), activation='relu'))
model2.add(MaxPooling2D(pool_size=(2,2)))
model2.add(Conv2D(128, (3,3), activation='relu'))
model2.add(MaxPooling2D(pool_size=(2,2)))
model2.add(Flatten())
model2.add(Dense(100, activation = "relu"))
model2.add(Dense(10, activation = "softmax"))
model2.summary()
```

```python
model2.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
start = time.time()
history2 = model2.fit(X_train, y_train, batch_size=32, epochs=25, validation_data=(X_validate, y_validate))
end = time.time()
totalTime = (end-start)/60
print("Total time: " + str(totalTime))

plt.plot(history2.history['accuracy'], color="blue", label="train")
plt.plot(history2.history['val_accuracy'], color="red", label="validate")
plt.xlabel("No of Epochs")
plt.ylabel("Accuracy")
plt.title("Accuracy plot for CNN #2")
plt.legend()
plt.show()

#create CNN 3 with 2 conv2d layers and L2 regularization

model3 = Sequential()

model3.add(Conv2D(128, (3,3), activation='relu',  input_shape=(32,32,3), kernel_regularizer=regularizers.l2(0.00001)))
model3.add(MaxPooling2D(pool_size=(2,2)))
model3.add(Conv2D(128, (3,3), activation='relu', kernel_regularizer=regularizers.l2(0.00001)))
model3.add(MaxPooling2D(pool_size=(2,2)))
model3.add(Flatten())
model3.add(Dense(100, activation = "relu", kernel_regularizer=regularizers.l2(0.00001)))
model3.add(Dense(10, activation = "softmax"))
model3.summary()

model3.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
start = time.time()
history3 = model3.fit(X_train, y_train, batch_size=32, epochs=25, validation_data=(X_validate, y_validate))
end = time.time()
totalTime = (end-start)/60
print("Total time: " + str(totalTime))

plt.plot(history3.history['accuracy'], color="blue", label="train")
plt.plot(history3.history['val_accuracy'], color="red", label="validate")
plt.xlabel("No of Epochs")
plt.ylabel("Accuracy")
plt.title("Accuracy plot for CNN #3")
plt.legend()
plt.show()

#create CNN 4 with 3 conv2d layers and L2 regularization

model4 = Sequential()

model4.add(Conv2D(128, (3,3), activation='relu', input_shape=(32,32,3), kernel_regularizer=regularizers.l2(0.00001)))
model4.add(MaxPooling2D(pool_size=(2,2)))
model4.add(Conv2D(128, (3,3), activation='relu', kernel_regularizer=regularizers.l2(0.00001)))
model4.add(MaxPooling2D(pool_size=(2,2)))
model4.add(Conv2D(128, (3,3), activation='relu',  kernel_regularizer=regularizers.l2(0.00001)))
model4.add(MaxPooling2D(pool_size=(2,2)))
model4.add(Flatten())
model4.add(Dense(100, activation = "relu", kernel_regularizer=regularizers.l2(0.00001)))
model4.add(Dense(10, activation = "softmax"))
model4.summary()

model4.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
start = time.time()
history4 = model4.fit(X_train, y_train, batch_size=32, epochs=25, validation_data=(X_validate, y_validate))
end = time.time()
totalTime = (end-start)/60
print("Total time: " + str(totalTime))

plt.plot(history4.history['accuracy'], color="blue", label="train")
plt.plot(history4.history['val_accuracy'], color="red", label="validate")
plt.xlabel("No of Epochs")
plt.ylabel("Accuracy")
plt.title("Accuracy plot for CNN #4")
plt.legend()
plt.show()

#create CNN 5 with 2 conv2d layers and Dropout regularization

model5 = Sequential()
```

```python
model5.add(Conv2D(128, (3,3), activation='relu', input_shape=(32,32,3)))
model5.add(MaxPooling2D(pool_size=(2,2)))
model5.add(Dropout(0.2))
model5.add(Conv2D(128, (3,3), activation='relu'))
model5.add(MaxPooling2D(pool_size=(2,2)))
model5.add(Dropout(0.2))
model5.add(Flatten())
model5.add(Dense(100, activation = "relu"))
model5.add(Dropout(0.2))
model5.add(Dense(10, activation = "softmax"))
model5.summary()

model5.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
start = time.time()
history5 = model5.fit(X_train, y_train, batch_size=32, epochs=25, validation_data=(X_validate, y_validate))
end = time.time()
totalTime = (end-start)/60
print("Total time: " + str(totalTime))

plt.plot(history5.history['accuracy'], color="blue", label="train")
plt.plot(history5.history['val_accuracy'], color="red", label="validate")
plt.xlabel("No of Epochs")
plt.ylabel("Accuracy")
plt.title("Accuracy plot for CNN #5")
plt.legend()
plt.show()

#create CNN 6 with 3 conv2d layers and Dropout regularization

model6 = Sequential()

model6.add(Conv2D(128, (3,3), activation='relu',input_shape=(32,32,3)))
model6.add(MaxPooling2D(pool_size=(2,2)))
model6.add(Dropout(0.2))
model6.add(Conv2D(128, (3,3), activation='relu'))
model6.add(MaxPooling2D(pool_size=(2,2)))
model6.add(Dropout(0.2))
model6.add(Conv2D(128, (3,3), activation='relu'))
model6.add(MaxPooling2D(pool_size=(2,2)))
model6.add(Dropout(0.2))
model6.add(Flatten())
model6.add(Dense(100, activation = "relu"))
model6.add(Dropout(0.2))
model6.add(Dense(10, activation = "softmax"))
model6.summary()

model6.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
start = time.time()
history6 = model6.fit(X_train, y_train, batch_size=32, epochs=25, validation_data=(X_validate, y_validate))
end = time.time()
totalTime = (end-start)/60
print("Total time: " + str(totalTime))

plt.plot(history6.history['accuracy'], color="blue", label="train")
plt.plot(history6.history['val_accuracy'], color="red", label="validate")
plt.xlabel("No of Epochs")
plt.ylabel("Accuracy")
plt.title("Accuracy plot for CNN #6")
plt.legend()
plt.show()

#create CNN 7 with 2 conv2d layers and Batch regularization

model7 = Sequential()

model7.add(Conv2D(128, (3,3), activation='relu', input_shape=(32,32,3)))
model7.add(MaxPooling2D(pool_size=(2,2)))
model7.add(BatchNormalization())
model7.add(Conv2D(128, (3,3), activation='relu'))
model7.add(MaxPooling2D(pool_size=(2,2)))
model7.add(BatchNormalization())
model7.add(Flatten())
model7.add(Dense(100, activation = "relu"))
model7.add(BatchNormalization())
model7.add(Dense(10, activation = "softmax"))
model7.summary()
```

```
model7.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
start = time.time()
history7 = model7.fit(X_train, y_train, batch_size=32, epochs=25, validation_data=(X_validate, y_validate))
end = time.time()
totalTime = (end-start)/60
print("Total time: " + str(totalTime))

plt.plot(history7.history['accuracy'], color="blue", label="train")
plt.plot(history7.history['val_accuracy'], color="red", label="validate")
plt.xlabel("No of Epochs")
plt.ylabel("Accuracy")
plt.title("Accuracy plot for CNN #7")
plt.legend()
plt.show()

#create CNN 8 with 3 conv2d layers and Batch regularization

model8 = Sequential()

model8.add(Conv2D(128, (3,3), activation='relu', input_shape=(32,32,3)))
model8.add(MaxPooling2D(pool_size=(2,2)))
model8.add(BatchNormalization())
model8.add(Conv2D(128, (3,3), activation='relu'))
model8.add(MaxPooling2D(pool_size=(2,2)))
model8.add(BatchNormalization())
model8.add(Conv2D(128, (3,3), activation='relu'))
model8.add(MaxPooling2D(pool_size=(2,2)))
model8.add(BatchNormalization())
model8.add(Flatten())
model8.add(Dense(100, activation = "relu"))
model8.add(BatchNormalization())
model8.add(Dense(10, activation = "softmax"))
model8.summary()

model8.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
start = time.time()
history8 = model8.fit(X_train, y_train, batch_size=32, epochs=25, validation_data=(X_validate, y_validate))
end = time.time()
totalTime = (end-start)/60
print("Total time: " + str(totalTime))

plt.plot(history8.history['accuracy'], color="blue", label="train")
plt.plot(history8.history['val_accuracy'], color="red", label="validate")
plt.xlabel("No of Epochs")
plt.ylabel("Accuracy")
plt.title("Accuracy plot for CNN #8")
plt.legend()
plt.show()

# adding training and validation data
x_train_final = np.concatenate([X_train, X_validate])
y_train_final = np.concatenate([y_train, y_validate])

model9 = Sequential()

model9.add(Conv2D(128, (3,3), activation='relu', input_shape=(32,32,3)))
model9.add(MaxPooling2D(pool_size=(2,2)))
#model9.add(BatchNormalization())
model9.add(Conv2D(128, (3,3), activation='relu'))
model9.add(MaxPooling2D(pool_size=(2,2)))
#model9.add(BatchNormalization())
#model9.add(Conv2D(128, (3,3), activation='relu'))
#model9.add(MaxPooling2D(pool_size=(2,2)))
#model9.add(BatchNormalization())
model9.add(Flatten())
model9.add(Dense(100, activation = "relu"))
#model9.add(BatchNormalization())
model9.add(Dense(10, activation = "softmax"))
model9.summary()

model9.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
start = time.time()
history9 = model9.fit(x_train_final, y_train_final, batch_size=32, epochs=25)
end = time.time()
totalTime = (end-start)/60
print("Total time: " + str(totalTime))
```

```
score = model9.evaluate(X_test,y_test, verbose=0)
print("Test loss: %.4f" % score[0])
print("Test accuracy: %.2f" % score[1])
```

```python
# imports

import tensorflow as tf
from keras.datasets import cifar10
import numpy as np
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from tensorflow.python.keras import Sequential
from tensorflow.python.keras.layers import Dense, Conv2D, Flatten, MaxPooling2D
from tensorflow.python.keras.backend import categorical_crossentropy
from keras import regularizers
from tensorflow.keras.layers import BatchNormalization, Dropout
import time
from matplotlib import pyplot as plt

# import ciraf10 image dataset and convert to train/validate/test dataset

(xTrain, yTrain), (xTest, yTest) = cifar10.load_data()
totalX = np.concatenate([xTrain, xTest])
totalY = np.concatenate([yTrain, yTest])
#first we split into 70% training data and 30% test data
X_train, X_test_temp, y_train, y_test_temp = train_test_split(totalX, totalY, test_size=0.3, random_state=1)
#then we split the test data into 50% of validate data making it 15% of the total
X_validate, X_test, y_validate, y_test = train_test_split(X_test_temp, y_test_temp, test_size=0.5, random_state=1)

# preprocess the data

X_train = X_train.reshape(X_train.shape[0], 32,32,3)
X_test = X_test.reshape(X_test.shape[0], 32,32,3)
X_validate = X_validate.reshape(X_validate.shape[0], 32,32,3)

y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
y_validate = to_categorical(y_validate)

# adding training and validation data
x_train_final = np.concatenate([X_train, X_validate])
y_train_final = np.concatenate([y_train, y_validate])

model9 = Sequential()

model9.add(Conv2D(128, (3,3), activation='relu', input_shape=(32,32,3)))
model9.add(MaxPooling2D(pool_size=(2,2)))
#model9.add(BatchNormalization())
model9.add(Conv2D(128, (3,3), activation='relu'))
model9.add(MaxPooling2D(pool_size=(2,2)))
#model9.add(BatchNormalization())
#model9.add(Conv2D(128, (3,3), activation='relu'))
#model9.add(MaxPooling2D(pool_size=(2,2)))
#model9.add(BatchNormalization())
model9.add(Flatten())
model9.add(Dense(100, activation = "relu"))
#model9.add(BatchNormalization())
model9.add(Dense(10, activation = "softmax"))
model9.summary()

model9.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
start = time.time()
history9 = model9.fit(x_train_final, y_train_final, batch_size=32, epochs=25)
end = time.time()
totalTime = (end-start)/60
print("Total time: " + str(totalTime))

score = model9.evaluate(X_test,y_test, verbose=0)
print("Test loss: %.4f" % score[0])
print("Test accuracy: %.2f" % score[1])

input = model9.input
output = model9.get_layer('conv2d_3').output
modelTest = Model(inputs = input, outputs=output)

from PIL import Image
from urllib import request
from io import BytesIO

url = "https://hips.hearstapps.com/hmg-prod.s3.amazonaws.com/images/2019-honda-civic-sedan-1558453497.jpg?crop=1xw:0.9997727789138833xh;center,top&resize=480:*"
res = request.urlopen(url).read()
#img = Image.open('car2.png').resize((32,32,3))
img = Image.open(BytesIO(res)).resize((32,32))
plt.imshow(img)

from keras.preprocessing.image import img_to_array
from keras.applications.vgg16 import preprocess_input

image = img_to_array(img)
image = np.expand_dims(image, axis = 0)
image = preprocess_input(image)
feature_maps = modelTest.predict(image)
square = 4
ix = 1
plt.figure(dpi = 200)
for _ in range(square):
  for _ in range(square):
    ax = plt.subplot(square, square, ix)
    ax.set_xticks([])
    ax.set_yticks([])
    plt.imshow(feature_maps[0, :, : ,ix-1], cmap = "gray")
    ix+=1

plt.show()

output2 = model9.get_layer('batch_normalization_4').output
modelTest2 = Model(inputs = input, outputs=output2)
feature_maps2 = modelTest2.predict(image)
```

```
square = 4
ix = 1
plt.figure(dpi = 200)
for _ in range(square):
  for _ in range(square):
    ax = plt.subplot(square, square, ix)
    ax.set_xticks([])
    ax.set_yticks([])
    plt.imshow(feature_maps2[0, :, : ,ix-1], cmap = "gray")
    ix+=1

plt.show()
```