# CIFAR-10 Image Classification with Explanations

Krupal Jadhav and Sheryl Mathew

***Abstract*—Image classification is one of the most important practical application of machine learning. Right from detecting pneumonia in x-ray images to classification of particle collisions in High Energy Physics, image classification has numerous applications. In this project, we propose a convolutional neural network model to classify images into different categories. The problem is a multiclass classification problem where we classify images into 10 different classes. Along with this we have performed hyperparameter tuning to find the best hyperparameters for our CNN model and have also used Local Interpretable Model-Agnostic Explanations (LIME) [1] to explain the predictions of our model.**

## I. Introduction

### *A. Motivation*

The increased use of technology right from clicking selfies to medical reports has led to a rapid increase in data especially images. Most of this data is not classified and thus no significant information can be obtained. Image classification is an emerging branch which has a huge scope to work on. Successive images of mother's womb help us in evaluating the baby's progress. Also, it is possible to detect abnormalities in organs. We can use these images to even predict the occurrence of any disease and so the patient can be saved from fatal complications. With the myriad amount of application that exists in this field, I chose to do project in image classification.

### *B. Challenges*

The biggest challenge in the domain of image classification is the size of the training images. The ImageNet challenge is a very popular competition which aims at building models which can accurately classify images given in their dataset. As per ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2017, there are 1,281,167 images in the training set. Building a machine learning model trained on a such huge number of training set will be a very difficult task as it will take hours to train. Hence, it is important to create an algorithm which is time efficient.

### *C. Problem Definition*

To build a machine learning model which can classify images into their appropriate classes and explain why the model made the prediction it made. We have assumed that the images will be 32 x 32 pixels RGB images.

### *D. Solution Approach*

The model that is built to perform image classification is Convolutional Neural Networks. CNN's have been developed with the assumption that the input will be images. This helps to reduce the number of parameters that are needed for images over using an artificial neural network. It has the ability of translation invariance i.e. no matter the position of an object in image the image will be classified successfully. Advantages includes the ability of automatic feature extraction and the predictions are very fast after training. Disadvantages include its inability to explain its decisions, training time is high and requires a large amount of training data.

## II. Review of Concepts

### *A. Convolutional Neural Network (CNN)*

Convolutional Neural Networks are similar to Artificial Neural Networks. The layers are arranged in three dimensions: width, height and depth since the CNN sees images as volumes. The neurons in one layer will not connect to all the neurons in the next layer instead only to a small part of it. The output of the network will be along the depth direction and represented as a single vector containing the probability scores. There are three different types of layers which include convolutional layers, pooling layers and fully connected layers. The stacked form of these layers forms CNN. This is shown in Figure 1 [2].

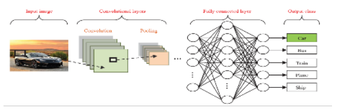


Fig. 1. Convolutional Neural Network

Convolutional layers are used to learn the feature representation of the given input image and act as feature extractors. A filter is passed over the input image and as it is being passed over the image, the values present in the filter are multiplied with the image pixels. After multiplying the values, they are summed to get a single number which is stored in the feature map. Then the filter is moved by the number of steps in stride and the process is repeated and the values stored in the feature/activation map. This process is called convolving. They are used to reducing the model complexity and can be optimized by varying the depth, stride and adding zero-padding.

Batch Normalization layers are used to adjust and scale the activations. This is done to reduce the amount of co variance shift. It allows each layer in the network to learn on its own and independently of the other layers.

Pooling layers are added to reduce the spatial dimensions of the input. This is done to reduce the number of parameters and hence the computational complexity. The pooling layer is applied over each feature/activation map independently.

Dropout layers are used to randomly set the input elements to zero with a given probability. This is done to prevent the network from over-fitting.

Fully connected layers are used to interpret the feature representations that have been generated by the convolutional and pooling layers and performs high-level reasoning. They consist of neurons which are connected to all the neurons before and after it without any layers in between them. This is similar to Artificial Neural Network.

## B. Keras

It is a high-level neural network API which is written in Python that performs deep learning functionalities. It supports both convolutional neural networks and recurrent neural networks. It runs on top of TensorFlow, Theano and PlaidML backends.

## C. Hyperparameter Tuning

*1) Hyperparameters:* Hyperparameters are those data which affects the model training. They are used to estimating the model parameters that will give the best predictions for the given data.

*2) Hyperparameter tuning principle:* Hyperparameter tuning is done by performing multiple trials in a single training. During each trial, the entire model to train is run using the values of the hyperparameters within the limits that had been set by the user. The model is optimized based on a metric that is selected by the user. When selecting the hyperparameters for the next trial, it tries to select the next those values which will improve the best result that had been achieved until then.

*3) Talos:* It is a hyperparameter scanning and optimization technique that is used in Keras. It uses grid, random and probabilistic hyperparameter optimization strategies which focus on maximizing the efficiency, flexibility as well as the result of random strategy. It also uses pseudo, quasi, true and quantum random methods. It's Prepare, Optimize and Deploy pipeline methodology helps increase its prediction capabilities.

## D. Activations

The input image which is fed to the CNN is displayed. The activations of the different layers of the network are visualized. The image activates the neurons of each convolutional layer and each filter is optimally activated for different features of the image. This allows users to see how the model transforms the image at each layer.

## E. Explainable Artificial Intelligence

*1) Overview:* It is an AI whose decisions can be trusted by humans. It gives an explanation for why a model took the decision that it took. This explanation of the decision-making process of a model allows users to determine whether the model has taken an accurate decision.

*2) Local Interpretable Model-Agnostic Explanations (LIME):* Lime [1]is a package which has the capability to explain any black box classifier with two or more classes. The classifier must take either a raw text or numpy array as an input and the output should be the probability for each class. The explanations are generated by approximating the given model by an interpretable model. The key idea is that it is easier to approximate a black-box model by a simple model *locally*. This is done by weighting the perturbed images by their similarity to the instance that needs to be explained.

## F. Evaluation Metrics

*1) Accuracy:* It is the ratio of number of correct predictions to the total number of input samples.
$Accuracy = \frac{Total\ number\ of\ correct\ predictions}{Number\ of\ examples}$

*2) Logarithmic Loss:* Logarithmic loss penalizes incorrect classification by assigning a probability for each class for all training examples. For M training examples and N number of classes, logarithmic loss can be calculated as,
$\log\ loss = \frac{-1}{M} \sum_j^M \sum_i^N y_{ij} \times \log(p_{ij})$

*3) Confusion Matrix:* A confusion matrix helps to analyze the following,

- True Positives (TP): Total positive examples classified as positive
- True Negatives (TN): Total negative examples classified as negative
- False Positives (FP): Total negative example classified as positive
- False Negatives (FN): Total positive examples classified as negative.

## III. Related Work

AlexNet [3] uses a deep convolutional neural network to classify images in the ILSVRC-2010 dataset. This brought deep learning into focus and is considered as one of the milestones in deep learning. It produced a top-5 test error rate of 15.4%. It contains 5 convolutional layers, 3 fully connected layers, dropout is applied before the first and second fully connected layer and Relu is applied after every convolutional and fully connected layer.

ZFNet [4] uses to classify images in the ImageNet 2012 dataset. It produced a top-5 test error rate of 14.8%. This was done by optimizing the hyper-parameters of AlexNet. Deconvolutional network was a visualization technique that was developed to examine the different feature activations and their relationships to the input.

GoogLeNet [5] uses a 22-layer deep CNN to classify images in the ILSVRC-2010 dataset. The network used batch normalization, RMSprop and image proportions to produce a top-5 test error rate of 6.67%. It consists of more than 100 layers and 20 parameter layers depth. It also used scale invariance and Hebbian principles. This network reduced the number of parameters from 60 million that was used in AlexNet to 4 million.

## IV. Experimental Setup

## A. Data

CIFAR-10 dataset consists of 60000 32x32 RGB images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The different classes pf images which are present include 'airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship' and 'truck'. The sample images from CIFAR-10 are shown in Figure 4.

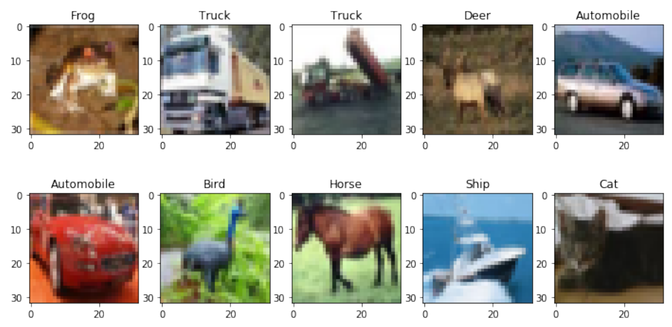


Fig. 2. Sample CIFAR-10 dataset images

### B. Model

The sequential convolutional neural network consists of the following:

- Two convolutional layers with filter size 32, 3x3 convolutional window and activation function relu. Each convolutional layer is followed by a batch normalization layer. This is the input layer
- A max pooling layer with a pool size 2x2 and stride 2 is added. This is followed by a dropout layer which drops 0.2 fraction of the inputs is added.
- Two convolutional layers with filter size 64, 3x3 convolutional window and activation function relu. Each convolutional layer is followed by a batch normalization layer.
- A max pooling layer with a pool size 2x2 and stride 2 is added. This is followed by a dropout layer which drops 0.3 fraction of the inputs is added.
- Two convolutional layers with filter size 128, 3x3 convolutional window and activation function relu. Each convolutional layer is followed by a batch normalization layer.
- A max pooling layer with a pool size 2x2 and stride 2 is added. This is followed by a dropout layer which drops 0.4 fraction of the inputs is added.
- A flatten layer is added to flatten all the inputs
- A dense layer which is a fully connected layer with 64 hidden units and activation function relu is added. This is followed by a dropout layer which drops 0.25 fraction of the inputs is added.
- A dense layer which is a fully connected layer with 32 hidden units and activation function relu is added. This is followed by a dropout layer which drops 0.25 fraction of the inputs is added.
- A dense layer which is a fully connected layer with 10 hidden units and activation function softmax is added. This is the output layer.

The optimization function which is used in the model is adam, the loss function which has to be minimized is categorical cross entropy and accuracy is the metric which will be evaluated by the model during training and testing.

### C. Hyperparameters

Some of the hyperparameters which are tuned in the CNN model include the following:

- Batch size: Number of samples per gradient update
- Number of epochs: Number of iterations over which the model is trained.
- Dropout fraction: The fraction of the inputs to be dropped
- Optimizer: Optimization function based on which the model weights are updated
- Loss function: Loss function to minimize
- Activation function for the convolutional layers: Function to decide whether a neuron in the model should be activated or node by calculating the weighted sum and adding the bias value to it.
- Activation function for the output layer: Function which decides the kind of output of each neuron in the model.

## V. Solution Approach

### A. Input

Since CIFAR-10 dataset already exists in the keras datasets we use the load_data () function to download it. We preprocess the images by reshaping them and normalizing the pixel values, so they lie between 0 and 1. The image labels are one-hot encoded. This preprocessing occurs for both training and testing data. The training images are split as 80% training images and 20% validation images.

### B. Without hyperparameter tuning

We have used an image data generator to augment to size of the images. Using a data generator will increase accuracy as the model will be trained on more number of images but it significantly affects the training time.

### C. With hyperparameter tuning

Talos is used for hyper-parameter tuning. The hyperparameters mentioned in Section 4.3 are given different ranges over which Talos will search over the model mentioned in Section 4.2 to find the best parameters for the given data. This model has no data generator.

The grid down sample value is set to 0.01. This s because otherwise the number of training iterations would have been 1600 which is very time-consuming. So, by setting the value to 0.01 we reduce the iterations to 16.

The scan summary details Figure 5 provides the details of the scan like the dimensions of the input image, dimensions of the input label, time at which the scanning completed, reduction metric among others.

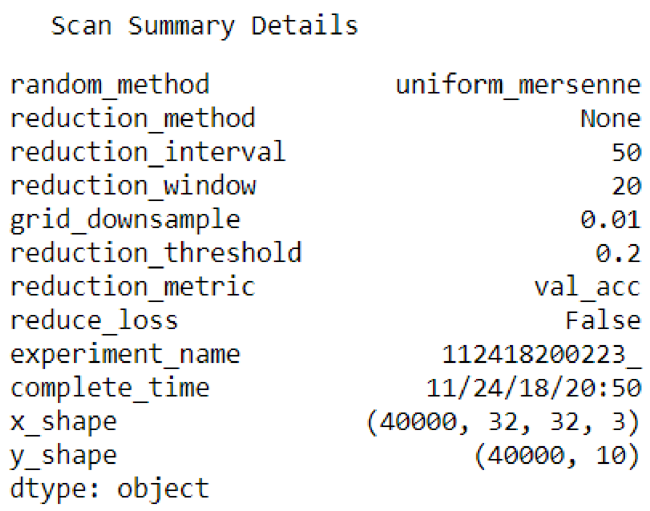
With the help of the reporting function of Talos, the hyperparameters which are selected in each iteration along with their training and validation accuracies are obtained. This can be seen in Figure 4 and Figure 5.

### D. Visualizations

A model with no data generator and no hyperparameter tuning is created as mentioned in Section 4.2.

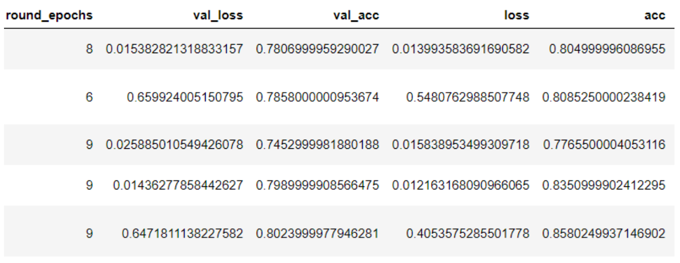
The different activation layers and their dimensions are shown below:

- conv2d_1/Relu:0(500, 32, 32, 32)



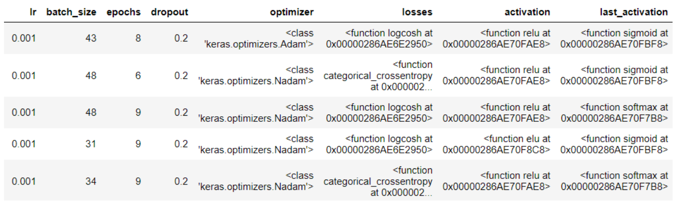
```
   Scan Summary Details

random_method              uniform_mersenne
reduction_method                       None
reduction_interval                       50
reduction_window                         20
grid_downsample                        0.01
reduction_threshold                     0.2
reduction_metric                    val_acc
reduce_loss                           False
experiment_name            112418200223_
complete_time               11/24/18/20:50
x_shape                 (40000, 32, 32, 3)
y_shape                        (40000, 10)
dtype: object
```

Fig. 3. Scan Summary Details



| round_epochs | val_loss | val_acc | loss | acc |
|---|---|---|---|---|
| 8 | 0.015382821318833157 | 0.7806999959290027 | 0.013993583691690582 | 0.804999996086955 |
| 6 | 0.659924005150795 | 0.7858000000953674 | 0.5480762988507748 | 0.8085250000238419 |
| 9 | 0.025885010549426078 | 0.7452999981880188 | 0.015838953499309718 | 0.7765500004053116 |
| 9 | 0.01436277858442627 | 0.7989999908566475 | 0.012163168090966065 | 0.8350999902412295 |
| 9 | 0.6471811138227582 | 0.8023999977946281 | 0.4053575285501778 | 0.8580249937146902 |

Fig. 4. Reporting Details - Loss and Accuracy



| lr | batch_size | epochs | dropout | optimizer | losses | activation | last_activation |
|---|---|---|---|---|---|---|---|
| 0.001 | 43 | 8 | 0.2 | <class 'keras.optimizers.Adam'> | <function logcosh at 0x00000286AE6E2950> | <function relu at 0x00000286AE70FAE8> | <function sigmoid at 0x00000286AE70FBF8> |
| 0.001 | 48 | 6 | 0.2 | <class 'keras.optimizers.Nadam'> | <function categorical_crossentropy at 0x000002... | <function relu at 0x00000286AE70FAE8> | <function sigmoid at 0x00000286AE70FBF8> |
| 0.001 | 48 | 9 | 0.2 | <class 'keras.optimizers.Nadam'> | <function logcosh at 0x00000286AE6E2950> | <function relu at 0x00000286AE70FAE8> | <function softmax at 0x00000286AE70F7B8> |
| 0.001 | 31 | 9 | 0.2 | <class 'keras.optimizers.Nadam'> | <function logcosh at 0x00000286AE6E2950> | <function elu at 0x00000286AE70F8C8> | <function sigmoid at 0x00000286AE70FBF8> |
| 0.001 | 34 | 9 | 0.2 | <class 'keras.optimizers.Nadam'> | <function categorical_crossentropy at 0x000002... | <function relu at 0x00000286AE70FAE8> | <function softmax at 0x00000286AE70F7B8> |

Fig. 5. Reporting Details - Selected hyperparameters

- batch_normalization_1/cond/Merge:0(500, 32, 32, 32)
- conv2d_2/Relu:0(500, 32, 32, 32)
- batch_normalization_2/cond/Merge:0(500, 32, 32, 32)
- max_pooling2d_1/MaxPool:0(500, 16, 16, 32)
- dropout_1/cond/Merge:0(500, 16, 16, 32)
- conv2d_3/Relu:0(500, 16, 16, 64)
- batch_normalization_3/cond/Merge:0(500, 16, 16, 64)
- conv2d_4/Relu:0(500, 16, 16, 64)
- batch_normalization_4/cond/Merge:0(500, 16, 16, 64)
- max_pooling2d_2/MaxPool:0(500, 8, 8, 64)
- dropout_2/cond/Merge:0(500, 8, 8, 64)
- conv2d_5/Relu:0(500, 8, 8, 128)
- batch_normalization_5/cond/Merge:0(500, 8, 8, 128)
- conv2d_6/Relu:0(500, 8, 8, 128)
- batch_normalization_6/cond/Merge:0(500, 8, 8, 128)
- max_pooling2d_3/MaxPool:0(500, 4, 4, 128)
- dropout_3/cond/Merge:0(500, 4, 4, 128)
- flatten_1/Reshape:0(500, 2048)
- dense_1/Relu:0(500, 64)
- dropout_4/cond/Merge:0(500, 64)
- dense_2/Relu:0(500, 32)
- dropout_5/cond/Merge:0(500, 32)
- dense_3/Softmax:0(500, 10)

Figure 6. shows the image of a cat and Figure 7. shows how the activation map of the image as it passes through the different layers.

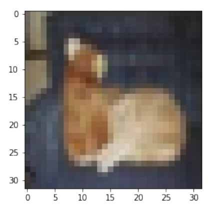


Fig. 6. Cat

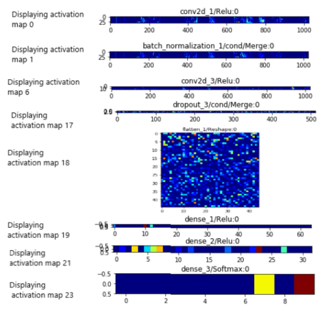


Fig. 7. Activation Map

Figure 8. shows the image of a truck and Figure 9 and 10. shows how that activations of the image as it passes through the different layers.

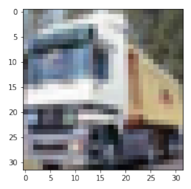


Fig. 8. Truck

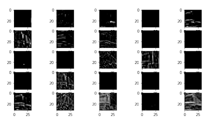


Fig. 9. Activations at layer 1

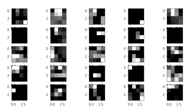


Fig. 10. Activations at layer 17

Figure 11. displays the heatmap for the confusion matrix.

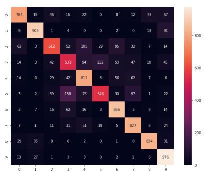


Fig. 11. Confusion Matrix

## E. *Explanations*

LIME [1] is used to explain the predictions of the model. Here we feed an image of an airplane and ask our model why it thinks it is an airplane and why it is not a bird.

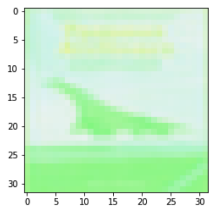


Fig. 12. Explanations for airplane

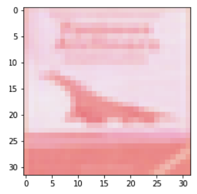


Fig. 13. Explanations for bird

Figure 12 is an explanation for an airplane whereas the figure 13 is an explanation for a bird. As we can see, the image is actually an airplane, and hence for the explanation for plane it completely masked the image with green, suggesting that all the features in the given image represent a plane whereas for explanation for bird, entire image was masked with red saying that none of the features in the image represented a bird.

## VI. Results

### A. *Without hyperparameter tuning and with data generator*

As data generator was a very slow process, we run our model for only 10 epochs. Table 1 displays the results we obtained for our run.

TABLE I
Evaluation Scores

| Data | Loss | Accuracy |
|---|---|---|
| Training set | 0.3773 | 0.8686 |
| Validation set | 0.3536 | 0.8888 |
| Test Set | 0.3911 | 0.8718 |

### B. *Without hyperparameter tuning and without data generator*

We run our CNN model for 100 epochs and obtained the results which are displayed in table 2.

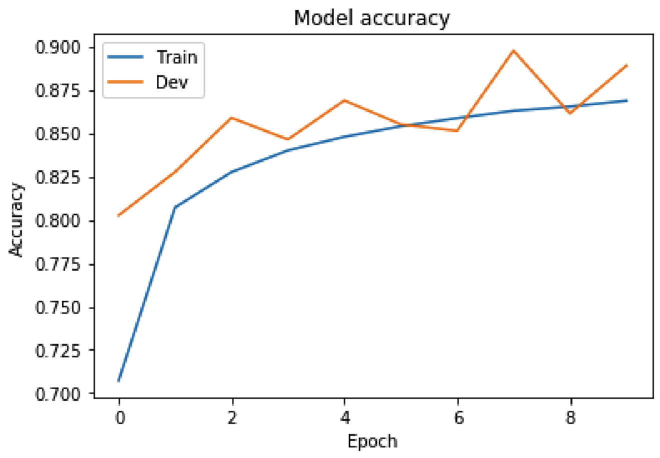


Fig. 14. Learning Curve

TABLE II
Evaluation Scores

| Data | Loss (100 epochs) | Accuracy (100 epochs) | Loss (10 epochs) | Accuracy (10 epochs) |
|---|---|---|---|---|
| Training set | 0.0217 | 0.9945 | 0.4217 | 0.6104 |
| Valida-tion set | 0.5821 | 0.8672 | 0.3996 | 0.6877 |
| Testing set | 0.5949 | 0.8114 | 0.4863 | 0.6394 |

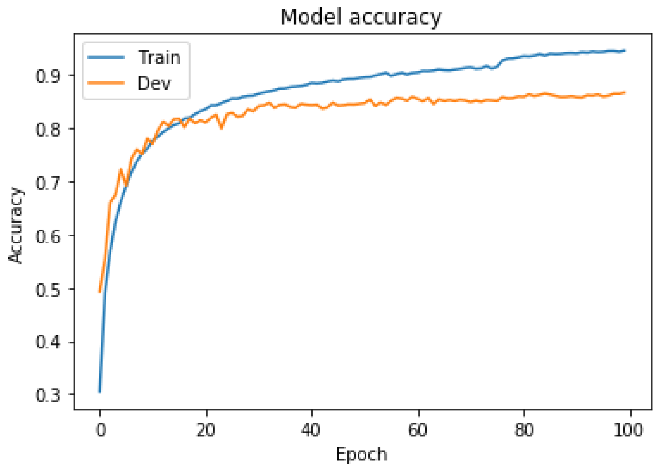


Fig. 15. Learning Curve

### C. *With hyperparameter tuning and without data generator*

Hyperparameter tuning was a slow process, so we performed hyperparameter tuning for only 10 epochs. Table 3 summarizes the results we obtained.

TABLE III
Evaluation Scores

| Data | Accuracy |
|---|---|
| Training set | 0.6253 after 10-fold cross validation |
| Validation set | 0.8056 |
| Test set | 0.7964 |

## VII. Lessons Learnt

We learnt a lot of lessons while doing this project. Some of these are,

- Neural network and convolutional neural network algorithms.
- Perform hyperparameter tuning in a convolutional neural network.
- Visualize the activations at different layers in a neural network.
- Explain the predictions of our model with the help of LIME [1].

## VIII. Conclusion and Future Scope

In this project we developed a convolutional neural network model to classify images into correct categories and for doing that we used CIFAR-10 dataset. We found out that performing hyperparameter tuning and augmenting our dataset significantly improved the accuracy of our model. We also saw how we can use LIME [1] to explain predictions of a CNN model. However, there is still room for more work in this project. As this is a very small dataset which has only 10 classes of image, one future scope can be to use a bigger dataset such as CIFAR-100 which consists of 100 classes. Also, due to hardware limitations, we did not complicate our model, so another work can be to complicate the model further by increasing number of epochs and adding more layers to improve the accuracy.

## References

[1] T. R. Marco, S. Sameer, and G. Carlos, “Why Should I Trust You¿‘: Explaining the Predictions of Any Classifier,” *CoRR*, vol. abs/1602.04938, 2016.

[2] W. Rawat and Z. Wang, “Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review,” *Neural Computation*, vol. 29, pp. 1–98, 06 2017. [Online]. Available: 10.1162/NECO_a_00990

[3] K. Alex, S. Ilya, and E. H. Geoffrey, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems*, p. 2012.

[4] M. D. Zeiler and R. Fergus, “Visualizing and Understanding Convolutional Networks,” in *Computer Vision - ECCV 2014*. Cham: Springer International Publishing, 2014, pp. 818–833.

[5] S. Christian, L. Wei, J. Yangqing, S. Pierre, R. Scott, A. Dragomir, E. Dumitru, V. Vincent, and R. Andrew, “Going Deeper with Convolutions,” in *Computer Vision and Pattern Recognition (CVPR)*, 2015.