

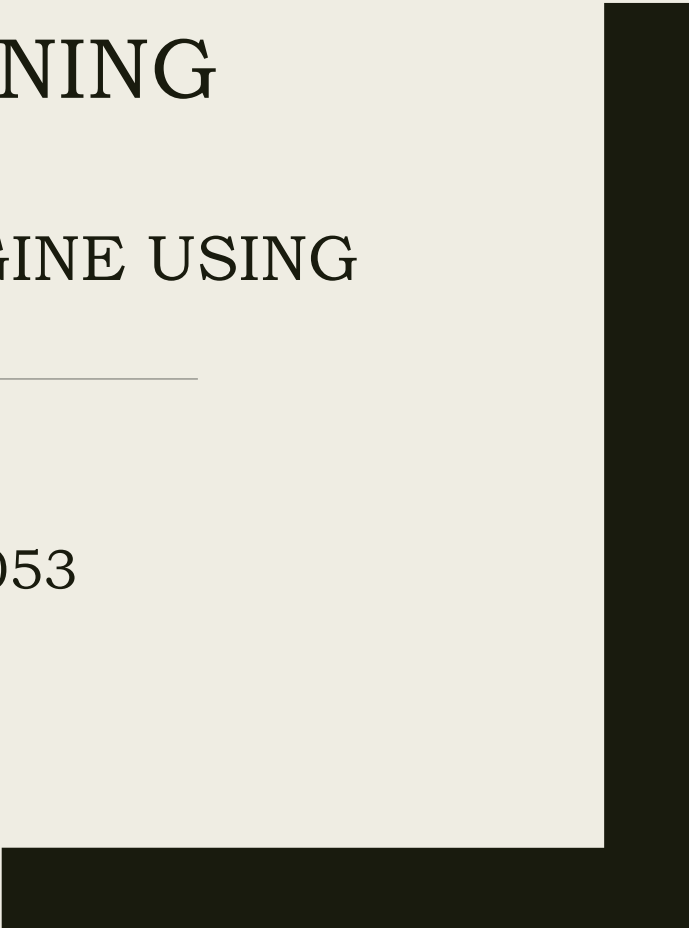


MINIPROJECT CS6301- MACHINE LEARNING

CIFAR-10 IMAGE RECOGNITION ENGINE USING CNNs

PRESENTED BY:
GOKUL . S – 2018103026
REUEL SAMUEL SAM – 2018103053

ON 6.11.2020



OBJECTIVE

- The modern internet world is heavily plagued with malicious and spam content that ruin the experience of a wonderful invention. One of the most prominent forms of such content is via images.
- Our objective is to identify and portray the features occurring in an image in terms of the object or type of land cover these features actually represent on the ground.
- This interest in image classification was piqued and we have tried to train a model that can classify images.
- For this purpose, we propose a convolutional neural network (CNN)-based architecture using a dataset of sufficient training capacity.

INTRODUCTION

- Image classification is an active research area and has been studied in popular applications such as driverless vehicles and emergency robots. Classification algorithms typically employ two phases of processing: **Training** and **Testing**.
- In the initial training phase, characteristic properties of typical image features are isolated and, based on these, a unique description of each classification category, *i.e. training class*, is created.
- In the subsequent testing phase, these feature-space partitions are used to classify image features.
- A CNN works by extracting features from images. This eliminates the need for manual feature extraction. The features are not trained! They're learned while the network trains on a set of images. This makes deep learning models extremely accurate for computer vision tasks. CNNs learn feature detection through tens or hundreds of hidden layers. Each layer increases the complexity of the learned features.

PROBLEM DESCRIPTION

- Type of Problem : Image Classification
- Dataset Used: CIFAR – 10
- Goal: Accurately classify unseen images using the trained model.
- Model Used: Convolutional Neural Network (CNN)
- Layers Used:
 - 6 x Convolutional Layers
 - 3 x Max Pooling Layers
 - 4 x Dropout Layers
 - 1 x Flatten Layer
 - 2 x Fully Connected Layers

MAIN CONCEPTS

■ ***Convolutional Neural Network***

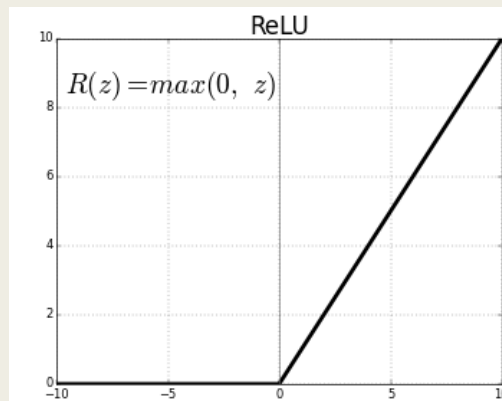
- Neural Network model with hierarchical layers
- Each layer involves convolutions with learned filters followed by pointwise non-linearity and downsampling operation called feature-pooling

■ ***Convolutional Layer***

- Parts of the image are taken and compared. These parts are called features/kernels.
- Each pixel is multiplied with its corresponding feature pixel. Find mean for that specific feature and replace the original pixel with this new value.

■ ***ReLU Activation***

- Rectified linear unit



■ ***Max-Pooling Layer***

- Used to shrink the image
- Specific Window size is chosen and the window is moved along the ReLU activated image.
- Maximum pixel value within the window is chosen to represent the group

■ ***Fully Connected Layer***

- Conversion of the resultant image into a flattened vector.

■ ***Dropout Layer***

- In Keras, dropout layer randomly sets input units to 0 during training while the inputs not set to 0 are scaled up. This is done to avoid the problem of over-fitting.

■ ***Batch Normalization***

- Standardizes the input at each layer for a mini-batch
- Standardization is the rescaling of data to have a mean of 0 and standard deviation of 1

Dataset Information

- CIFAR-10 is a labelled dataset consisting of numerous images of 32x32 dimensions classified and labelled into 10 different classes.
- The target classes are: Airplane, Automobile, Bird, Cat, Deer, Dog, Frog, Horse, Ship, Truck
- The classes are completely mutually exclusive. There is no overlapping between classes.
- This essentially means that there is no image that can be classified into two target classes. There are 60000 images with 10000 images in each class, making this a very large dataset to work on and use for training.
- Link: <https://www.cs.toronto.edu/~kriz/cifar.html>

System Architecture

Model Summary:

6 x Convolutional Layers

3 x Max Pooling Layers

4 x Dropout Layers

1 x Flatten Layer

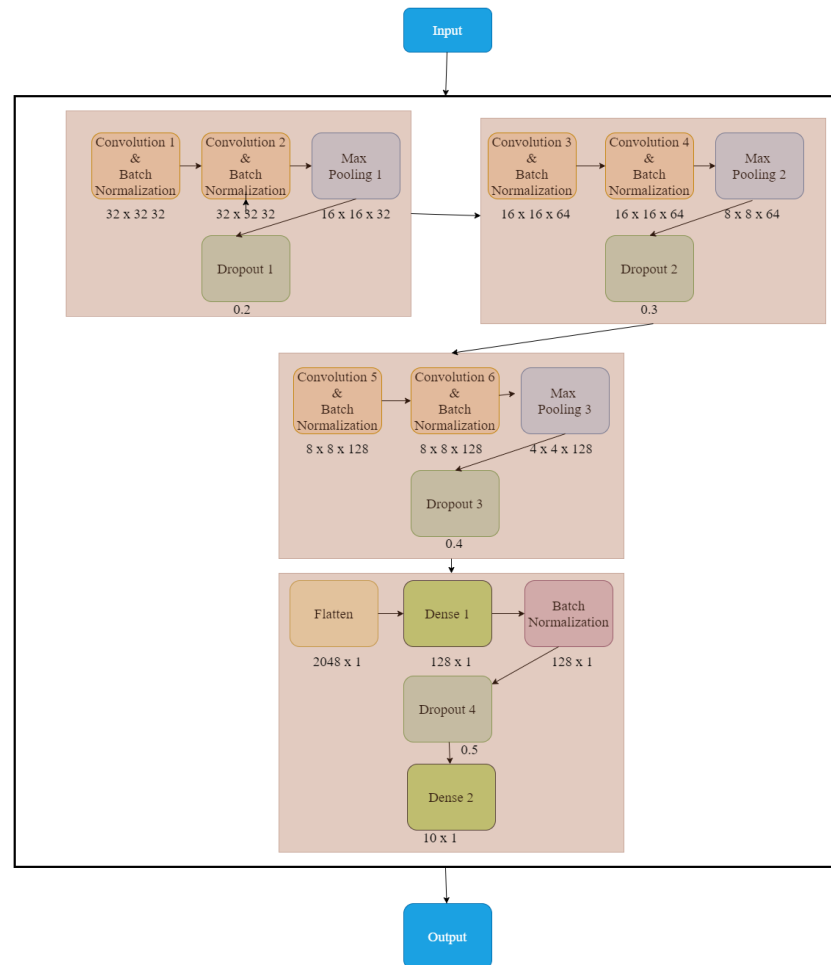
2 x Fully Connected Layers

Diagram follows in next slide

Model: "sequential"		
Layer (type)	Output Shape	Param #
=====		
conv2d (Conv2D)	(None, 32, 32, 32)	896
batch_normalization (Batch Normalization)	(None, 32, 32, 32)	128
conv2d_1 (Conv2D)	(None, 32, 32, 32)	9248
batch_normalization_1 (Batch Normalization)	(None, 32, 32, 32)	128
max_pooling2d (MaxPooling2D)	(None, 16, 16, 32)	0
dropout (Dropout)	(None, 16, 16, 32)	0
conv2d_2 (Conv2D)	(None, 16, 16, 64)	18496
batch_normalization_2 (Batch Normalization)	(None, 16, 16, 64)	256
conv2d_3 (Conv2D)	(None, 16, 16, 64)	36928
batch_normalization_3 (Batch Normalization)	(None, 16, 16, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 64)	0
dropout_1 (Dropout)	(None, 8, 8, 64)	0
conv2d_4 (Conv2D)	(None, 8, 8, 128)	73856
batch_normalization_4 (Batch Normalization)	(None, 8, 8, 128)	512
conv2d_5 (Conv2D)	(None, 8, 8, 128)	147584
batch_normalization_5 (Batch Normalization)	(None, 8, 8, 128)	512
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 128)	0
dropout_2 (Dropout)	(None, 4, 4, 128)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 128)	262272
batch_normalization_6 (Batch Normalization)	(None, 128)	512
dropout_3 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 10)	1290
=====		
Total params: 552,874		
Trainable params: 551,722		
Non-trainable params: 1,152		

System Architecture

GUI



Module-Wise Algorithms

■ Pre-defined modules used:

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
```

- *Tensorflow*
 - Open source Machine Learning Platform
- *Keras*
 - Open source library that provides python interface for a artificial neural networks
- *Matplotlib* (not necessarily required for functioning)
- *numpy*

■ Importing of Dataset:

```
global train_images
global train_labels
global test_images
global test_labels

(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()

# Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0

class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

print("Dataset loading Done")
```

- Dataset is loaded in from tensor flow library.
- Images are split and stored into training and testing data
- The images consist of pixels that have values between 0 to 255. We have divided each by 255 to achieve some form of normalization
- The different class labels are added to the list for convenient output (seen later)

Module

Snippet

Initialization

```
class CNN():  
  
    def __init__(self):  
        pass  
  
    def model_start(self):  
        self.model_create()  
        self.model_summary()  
        var = self.model_compile()  
        print("Model Creation Done")
```

Model Creation and Summary

NOTE: *Has been explained and expanded upon earlier*

```
def model_create(self):  
  
    self.model = Sequential()  
    self.model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same',  
input_shape=(32, 32, 3)))  
    self.model.add(BatchNormalization())  
    self.model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))  
    self.model.add(BatchNormalization())  
    self.model.add(MaxPool2D((2, 2)))  
    self.model.add(Dropout(0.2))  
    self.model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))  
    self.model.add(BatchNormalization())  
    self.model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))  
    self.model.add(BatchNormalization())  
    self.model.add(MaxPool2D((2, 2)))  
    self.model.add(Dropout(0.3))  
    self.model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform',  
padding='same'))  
    self.model.add(BatchNormalization())  
    self.model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform',  
padding='same'))  
    self.model.add(BatchNormalization())  
    self.model.add(MaxPool2D((2, 2)))  
    self.model.add(Dropout(0.4))  
    self.model.add(Flatten())  
    self.model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))  
    self.model.add(BatchNormalization())  
    self.model.add(Dropout(0.5))  
    self.model.add(Dense(10, activation='softmax'))  
  
def model_summary(self):  
    self.model.summary()
```

Module	Snippet
Compiling and Training	<pre>def model_compile(self): self.model.compile(optimizer='adam', loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy']) return 0 def train(self): global train_images global train_labels global test_images global test_labels self.history = self.model.fit(train_images, train_labels, epochs=50, validation_data=(test_images, test_labels))</pre>
Evaluation and Prediction of New	<pre>def model_evaluate(self): test_loss, test_acc = self.model.evaluate(test_images, test_labels, verbose=2) print(test_acc) return test_acc def predict_new(self, image_path): img=mpimg.imread(image_path) ret_class = class_names[self.model.predict(np.expand_dims(img,axis=0)).argmax()] print(ret_class) return ret_class</pre>

Module	Snippet
Saving and Loading of Model	<pre>def model_save(self): self.model.save('model') def model_load(self): self.model = tf.keras.models.load_model('model')</pre>

Additional Information

- **he_uniform Kernel Initializer**

- Initializes the kernel/feature by drawing samples from a uniform distribution

- **Sparse Categorical Cross Entropy**

- Cross entropy function used as in multi-class situations
- Cross Entropy is the measure of difference between two probability distributions for a given random variable.
- This is used as the loss function.

Invoking of CNN within GUI

- Initialization and training

```
def train_button_pressed(self, window):  
    global cifar  
    cifar = CNN_Cifar.CNN()  
  
    cifar.model_load()  
  
    msg2 = QMessageBox()  
    msg2.setWindowTitle("Training Complete ")  
    msg2.setText("Model has been loaded!")  
  
    msg2.setIcon(QMessageBox.Information)  
  
    msg2.setDefaultButton(QMessageBox.Ok)  
    msg2.exec_()  
    self.openWindow(2, window)
```

- Evaluation and Accuracy

```
def accuracy_button_pressed(self):  
    global cifar  
  
    acc = cifar.model_evaluate()  
    cifar.model_performance_graph()  
    msg1 = QMessageBox()  
    msg1.setWindowTitle("Accuracy ")  
    msg1.setText("Accuracy of trained model is: "+str(acc*100)+"%")  
    msg1.setIcon(QMessageBox.Information)  
    msg1.setDefaultButton(QMessageBox.Ok)  
    msg1.exec_()
```

■ New Image Classification

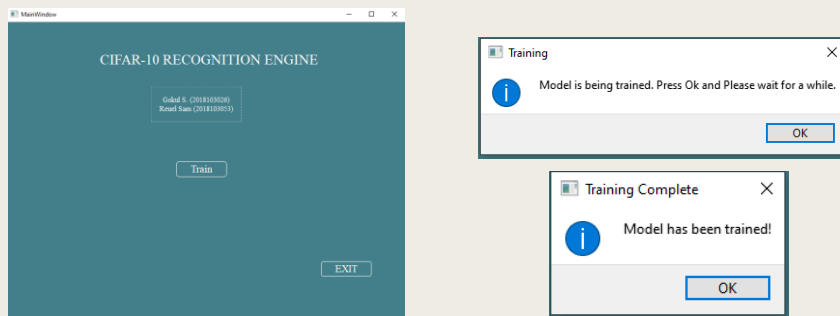
```
def submit_button_pressed(self):  
    global cifar  
    ret_class = cifar.predict_new(self.image_path)  
  
    msg1 = QMessageBox()  
    msg1.setWindowTitle("Result: ")  
    msg1.setText("The provided image has been recognized as '"+ret_class+"'")  
    msg1.setIconPixmap(QPixmap(self.image_path))  
    msg1.setDefaultButton(QMessageBox.Ok)  
    msg1.exec_()
```


Training

After the loading of the data and preprocessing, the model has to be trained.

The training took about 20-21 minutes.

Initially we had the training to be live when the model was much simpler as shown below:



```
In [6]: 1 model.train()

Epoch 44/50
50000/50000 [=====] - 23s 460us/sample - loss: 1.6249 - accuracy: 0.8353 - val_loss: 1.6301 - val_accuracy: 0.8303
Epoch 44/50
50000/50000 [=====] - 26s 519us/sample - loss: 1.6249 - accuracy: 0.8356 - val_loss: 1.6374 - val_accuracy: 0.8230
Epoch 45/50
50000/50000 [=====] - 25s 499us/sample - loss: 1.6247 - accuracy: 0.8356 - val_loss: 1.6796 - val_accuracy: 0.7796
Epoch 46/50
50000/50000 [=====] - 25s 499us/sample - loss: 1.6246 - accuracy: 0.8356 - val_loss: 1.6260 - val_accuracy: 0.8340
Epoch 47/50
50000/50000 [=====] - 25s 497us/sample - loss: 1.6241 - accuracy: 0.8365 - val_loss: 1.6310 - val_accuracy: 0.8294
Epoch 48/50
50000/50000 [=====] - 25s 498us/sample - loss: 1.6222 - accuracy: 0.8377 - val_loss: 1.6292 - val_accuracy: 0.8306
Epoch 49/50
50000/50000 [=====] - 37s 733us/sample - loss: 1.6222 - accuracy: 0.8380 - val_loss: 1.6309 - val_accuracy: 0.8301
Epoch 50/50
50000/50000 [=====] - 26s 525us/sample - loss: 1.6226 - accuracy: 0.8381 - val_loss: 1.6367 - val_accuracy: 0.8241

In [7]: 1 model.model_evaluate()

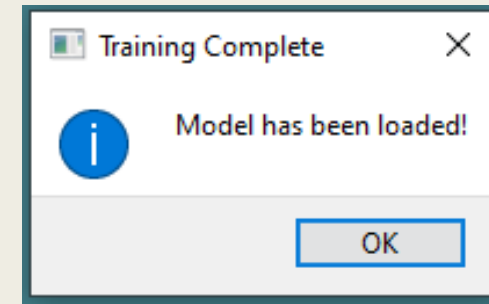
10000/10000 - 1s - loss: 1.6367 - accuracy: 0.8241
0.8241

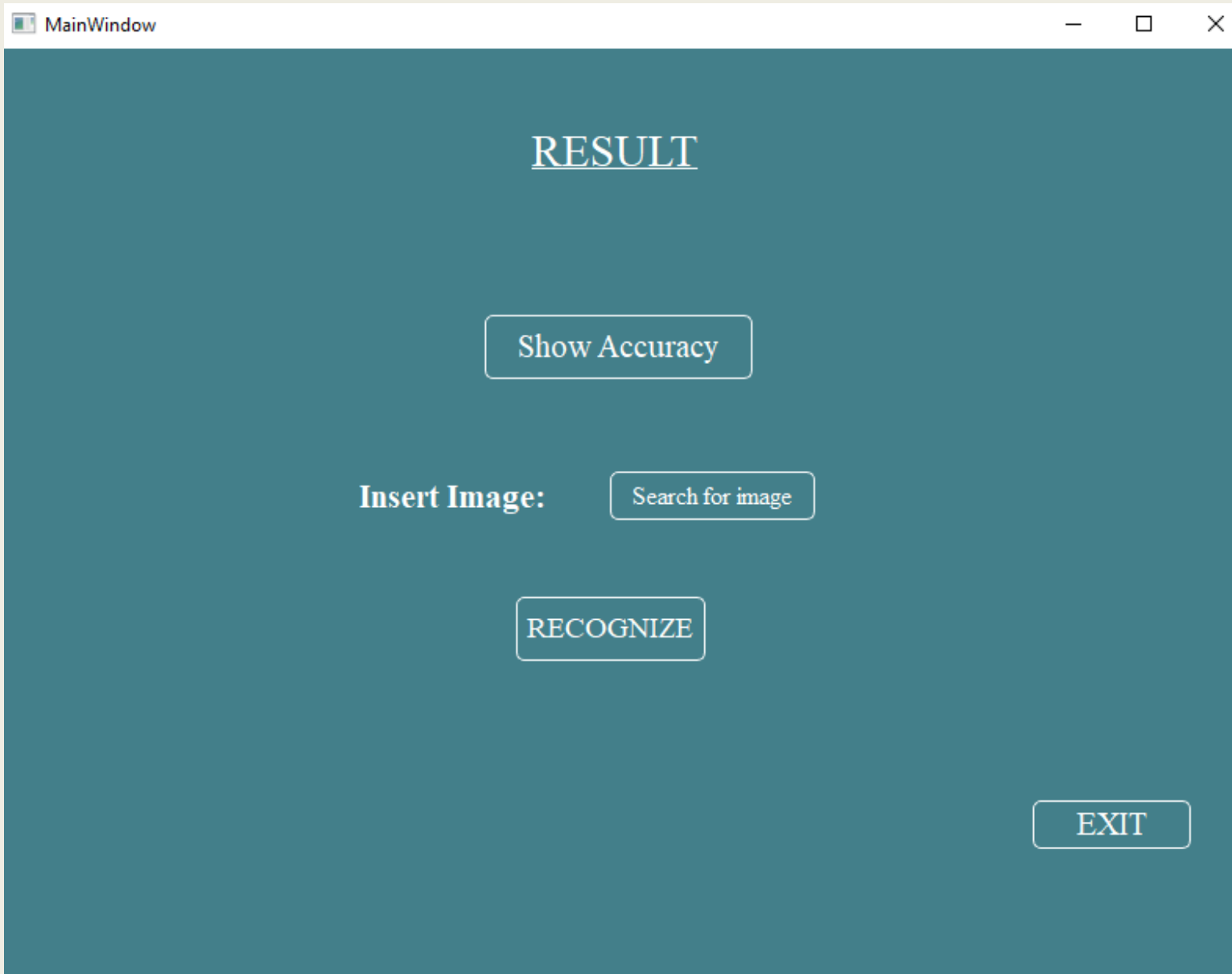
0.8241
```

This was then replaced by just loading of data as per instruction

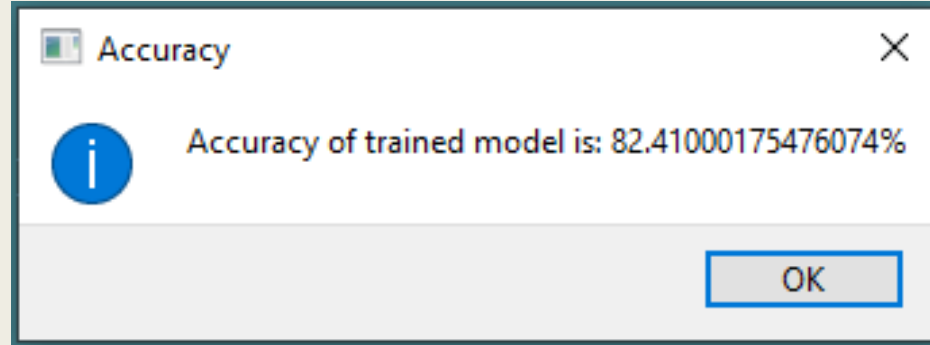
Shown in the next few slides

Output Screenshot





- Loss Function : Sparse Categorical Cross Entropy
- Performance Metric used: Sparse Categorical Accuracy



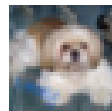
Insert Image:

Search for image

RECOGNIZE




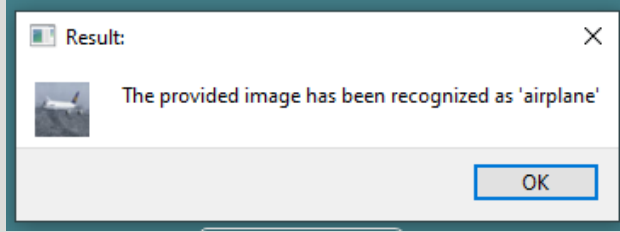

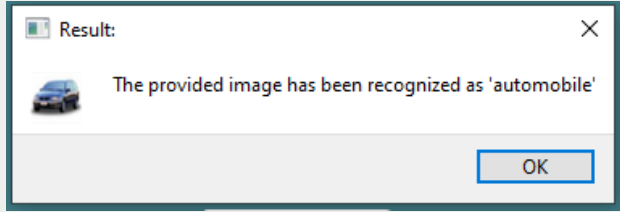

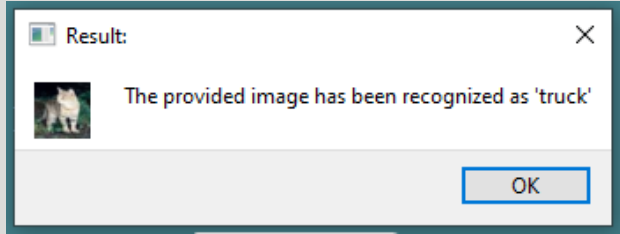
Result:




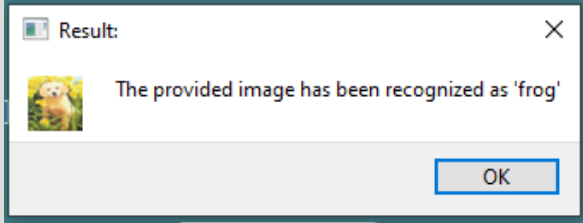

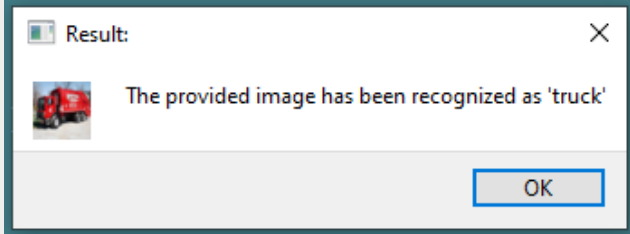
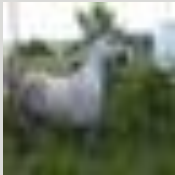
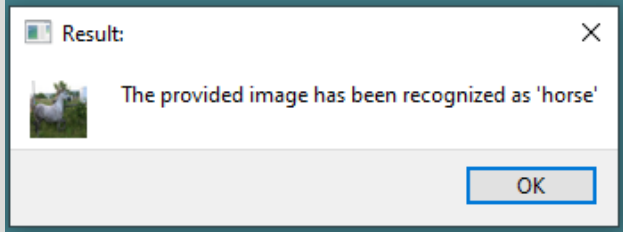
The provided image has been recognized as 'dog'

OK

Test Cases

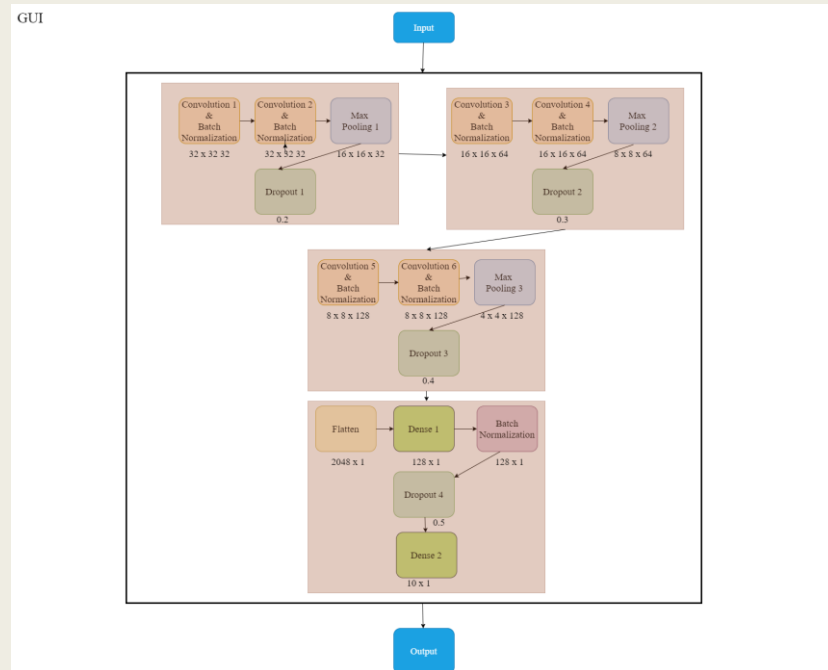
Input Image	Expected Classification	Actual Classification	Status
	Airplane		Correct
	Automobile		Correct
	Cat		Incorrect

Test Cases

Input Image	Expected Classification	Actual Classification	Status
	Dog		Incorrect
	Truck		Correct
	Horse		Correct

Analytical Ablation Study

- An analytical ablation study refers to the addition or removal of some features, layers, fine-tuning parameters etc. from the model and noting the difference in performance.
- This is done in order to understand the importance of a particular feature towards the final testing accuracy.
- **Original Model Architecture:**



Iteration 1:

```
Model: "sequential"
```

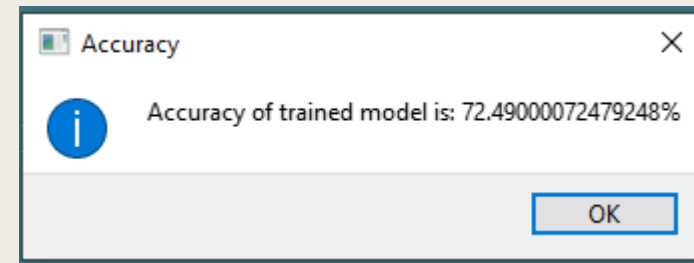
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_1 (Conv2D)	(None, 13, 13, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_2 (Conv2D)	(None, 4, 4, 64)	36928
flatten (Flatten)	(None, 1024)	0
dense (Dense)	(None, 64)	65600
dense_1 (Dense)	(None, 10)	650

```
Total params: 122,570  
Trainable params: 122,570  
Non-trainable params: 0
```

Removed:

- Batch Normalizations
- 3 Convolution Layers
- 1 Max-Pooling Layer
- All Dropouts
- 1 Dense Layer

- Accuracy obtained for this model:



Iteration 2:

```
Model: "sequential"
```

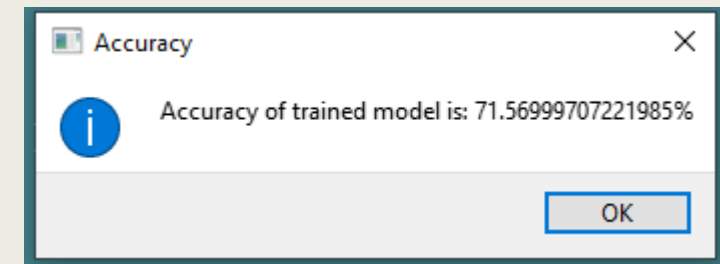
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_1 (Conv2D)	(None, 13, 13, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_2 (Conv2D)	(None, 4, 4, 128)	73856
max_pooling2d_2 (MaxPooling2D)	(None, 2, 2, 128)	0
flatten (Flatten)	(None, 512)	0
dense (Dense)	(None, 128)	65664
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 64)	8256
dropout_1 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 10)	650

```
Total params: 167,818  
Trainable params: 167,818  
Non-trainable params: 0
```

Added:

- 1 New Max-Pooling Layer
- 2 Dropout Layers
- 1 New Dense Layer
- Edited dimensions of Conv2D layers

- Accuracy obtained for this model:



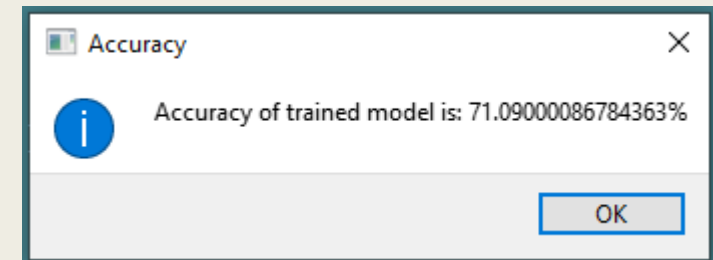
Iteration 3:

```
Model: "sequential"
Layer (type)                 Output Shape              Param #
=====
conv2d (Conv2D)              (None, 30, 30, 32)       896
max_pooling2d (MaxPooling2D) (None, 15, 15, 32)       0
conv2d_1 (Conv2D)            (None, 13, 13, 64)       18496
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64)         0
conv2d_2 (Conv2D)            (None, 4, 4, 128)        73856
max_pooling2d_2 (MaxPooling2 (None, 2, 2, 128)        0
flatten (Flatten)            (None, 512)              0
dense (Dense)                (None, 128)              65664
dense_1 (Dense)              (None, 64)               8256
dense_2 (Dense)              (None, 10)               650
=====
Total params: 167,818
Trainable params: 167,818
Non-trainable params: 0
```

Removed:

- 2 Dropout layers

- Accuracy obtained for this model:



Iteration 4:

Model: "sequential"

Layer (type)	Output Shape	Param #
=====		
conv2d (Conv2D)	(None, 30, 30, 32)	896

max_pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0

conv2d_1 (Conv2D)	(None, 13, 13, 64)	18496

max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 64)	0

conv2d_2 (Conv2D)	(None, 4, 4, 128)	73856

max_pooling2d_2 (MaxPooling2D)	(None, 2, 2, 128)	0

flatten (Flatten)	(None, 512)	0

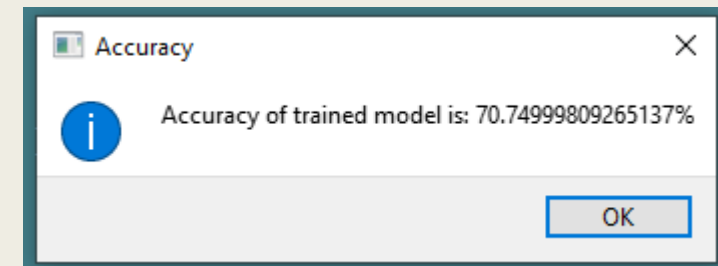
dense (Dense)	(None, 64)	32832

dense_1 (Dense)	(None, 10)	650
=====		
Total params: 126,730		
Trainable params: 126,730		
Non-trainable params: 0		

Removed:

- 1 Dense layer

- Accuracy obtained for this model:



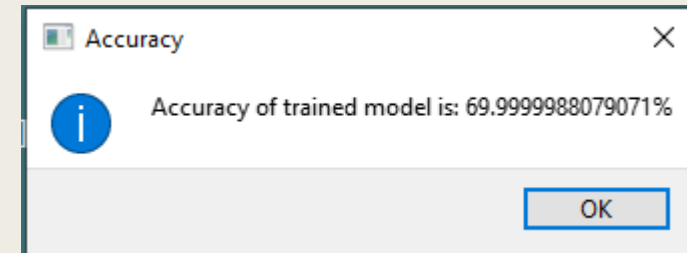
Iteration 5:

```
Model: "sequential"
Layer (type)                Output Shape              Param #
-----
conv2d (Conv2D)             (None, 30, 30, 32)       896
max_pooling2d (MaxPooling2D) (None, 15, 15, 32)       0
conv2d_1 (Conv2D)           (None, 13, 13, 64)       18496
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64)       0
conv2d_2 (Conv2D)           (None, 4, 4, 128)        73856
max_pooling2d_2 (MaxPooling2 (None, 2, 2, 128)       0
flatten (Flatten)           (None, 512)              0
dense (Dense)               (None, 10)               5130
-----
Total params: 98,378
Trainable params: 98,378
Non-trainable params: 0
```

Removed from:

- 1 Dense layer

- Accuracy obtained for this model:



Analytical Ablation Study: Inferences

- Normalization of batches heavily increases accuracy of the model
- Model performs better with dropout layers as it prevents over-fitting (**Dropout Regularization**).
- Removing Dense Layers does not affect the model by a significant amount. Validation accuracy does not deviate by much.
- This implies that most of the significant performance of a CNN is attributed to the Convolutional layers and not the fully connected Dense Layers.
- This can be seen from the fact that the accuracy has not varied much from Iteration 3 through Iteration 5.

Conclusion

- As seen in the prior slides, we were able to achieve the goals we set.
- The model was successfully able to classify unseen images correctly.
- We were able to make a sophisticated application that allowed the user to interact with the image classification engine in an effortless manner.
- The engine developed can further be used in numerous other applications including captcha tests, malicious content classification etc.

References

- Sharma, Neha & Jain, Vibhor & Mishra, Anju. (2018). An Analysis Of Convolutional Neural Networks For Image Classification. Procedia Computer Science. 132. 377-384. 10.1016/j.procs.2018.05.198
- Calık, Caner & Demirci, M. Fatih. (2018). Cifar-10 Image Classification with Convolutional Neural Networks for Embedded Systems. 1-2. 10.1109/AICCSA.2018.8612873
- Simonyan, Karen & Zisserman, Andrew. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv 1409.1556.