DAYANANDA SAGAR UNIVERSITY

KUDLU GATE, BANGALORE – 560068

Bachelor of Technology
in
CSE(Artificial Intelligence and Machine Learning)

Deep Learning

(CIFAR-10 Image Recognition using ResNet-50 model)

By

Akshay Hegde(ENG20AM0006)
Darshan R(ENG20AM0019)
M Adarsh Pryan(ENG20AM0034)
Sneha Subramaniam(ENG20AM0054)

Under the supervision of

Dr.Jayavrinda Vrindavanam

Professor and Chairperson CSE(AI&ML) Department

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING,
SCHOOL OF ENGINEERING
DAYANANDA SAGAR UNIVERSITY,
BANGALORE
(2022-2023)

MINOR PROJECT REPORT

Content

1.ABSTRACT

The CIFAR-10 dataset is a widely used benchmark dataset in deep learning and computer vision research. The dataset consists of 60,000 32x32 color images in 10 classes, with 6,000 images per class. The 10 classes are: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.

Deep learning models such as convolution neural networks have been successful in image classification and object detection tasks. Cifar-10 dataset is used in this paper to benchmark our deep learning model. Various function optimization methods such as Adam, RMS along with various regularization techniques are used to get good accuracy on image classification task.

2.INTRODUCTION

The CIFAR-10 dataset (Canadian Institute For Advanced Research) is a collection of images that are commonly used to train machine learning and computer vision algorithms. It is one of the most widely used datasets for machine learning research. The CIFAR-10 dataset contains 60,000 32x32 color images in 10 different classes

The 10 different classes represent airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. There are 6,000 images of each class

Computer algorithms for recognizing objects in photos often learn by example, CIFAR-10 is a set of images that can be used to teach a computer how to recognize objects.

Since the images in CIFAR-10 are low-resolution (32x32), this dataset can allow researchers to quickly try different algorithms works. Various kinds of convolutional neural networks tend to be the best at recognizing the images in CIFAR-10. • CIFAR-10 is a labeled subset of the 80 million tiny images dataset. When the dataset was created, students were paid to label all of the images.

(fig 2.1 : CIFAR-10 DATASET)

3.PROBLEM STATEMENT

The problem statement of a CIFAR-10 deep learning project is typically to train a model that can accurately classify the 10 different classes of images in the dataset. The goal is to develop a deep learning model that can generalize well to new, unseen images and achieve high accuracy on the test set.

The task of image classification in CIFAR-10 is a challenging problem due to the relatively lowresolution of the images and the small size of the objects in the images. Additionally, some of the classes, such as "bird" and "dog", have significant intra-class variability, making it difficult to accurately distinguish between different instances of these classes.

Therefore, the main objective of a CIFAR-10 deep learning project is to develop a model that can effectively learn discriminative features from the input images and use these features to accurately classify the images into the correct classes. The success of such a project is typically evaluated based on the accuracy of the model on the test set.

(fig 3.1: CIFAR-10 DATASET)

4.LITERATURE REVIEW

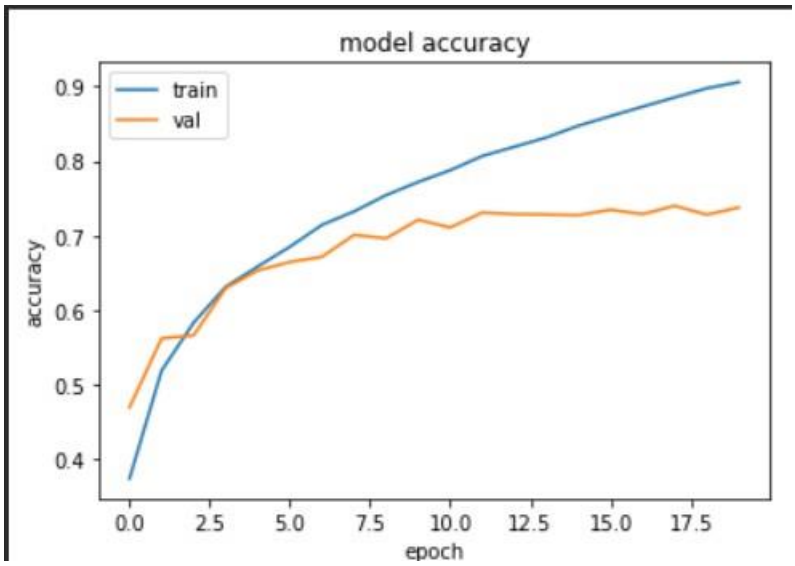
Many researchers have used the dataset to evaluate and compare the performance of various deep learning models and techniques. One commonly used model for CIFAR-10 classification is the Convolutional Neural Network (CNN). In particular, the use of deeper CNN architectures has been shown to improve the accuracy of CIFAR-10 classification. For example, the VGG-16 architecture achieved state-of-the-art performance on CIFAR-10 by incorporating deeper convolutional layers.

Another popular model for CIFAR-10 classification is the Residual Network (ResNet). ResNet architectures have been shown to be highly effective for image recognition tasks, and have achieved excellent performance on CIFAR-10, with ResNet-164 achieving state-of-the-art performance on the dataset.

In addition to architecture design, there has been significant research on optimization techniques for improving the performance of deep learning models on CIFAR-10. For example, researchers have explored the use of different activation functions, weight initialization strategies, and regularization techniques, among others.

Finally, there has been some work on using CIFAR-10 as a transfer learning dataset, by pre-training models on the larger ImageNet dataset and fine-tuning them on CIFAR-10. This approach has been shown to be effective in achieving high accuracy on CIFAR-10 while reducing the amount of training time required.

Dataset	Network	GFLOPs	Acc(%)
CIFAR-10	ResNet101 [35]	2.50	93.75
	Res50 [35]	1.29	93.62
	Res18 [35]	0.55	93.02
	IamNN [31]	1.10	94.60
	DG-Res [28] - config A	2.22	91.99
	DG-Res [28] - config B	2.82	92.97
	DG-Res [28] - config C	3.20	93.99
	DG-Res [28] - config D	3.93	94.70
	CGap [43]	> 0.87	93.20
	ResNet-110-pruned [44]	2.13	93.55
	ResNet-56-pruned [44]	0.91	93.06
	LC-Net	**0.33**	**95.25**
	LC-Net pre-trained (parallel)	**0.19**	**93.27**
	LC-Net pre-trained (sequential)	**0.06**	**93.27**



(fig 4.1: Plot of accuracy vs epoch of the model)

4.1. Project Description

The CIFAR-10 dataset is a widely used benchmark dataset in deep learning and computer vision research. It consists of 60,000 32x32 color images in 10 classes, with 6,000 images per class. The 10 classes are: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.

The dataset is divided into two sets: a training set with 50,000 images and a test set with 10,000 images. The training set is used to train the deep learning model, while the test set is used to evaluate the performance of the trained model.

The goal of this Project is to classify different images into respective classes.

5.Methodology

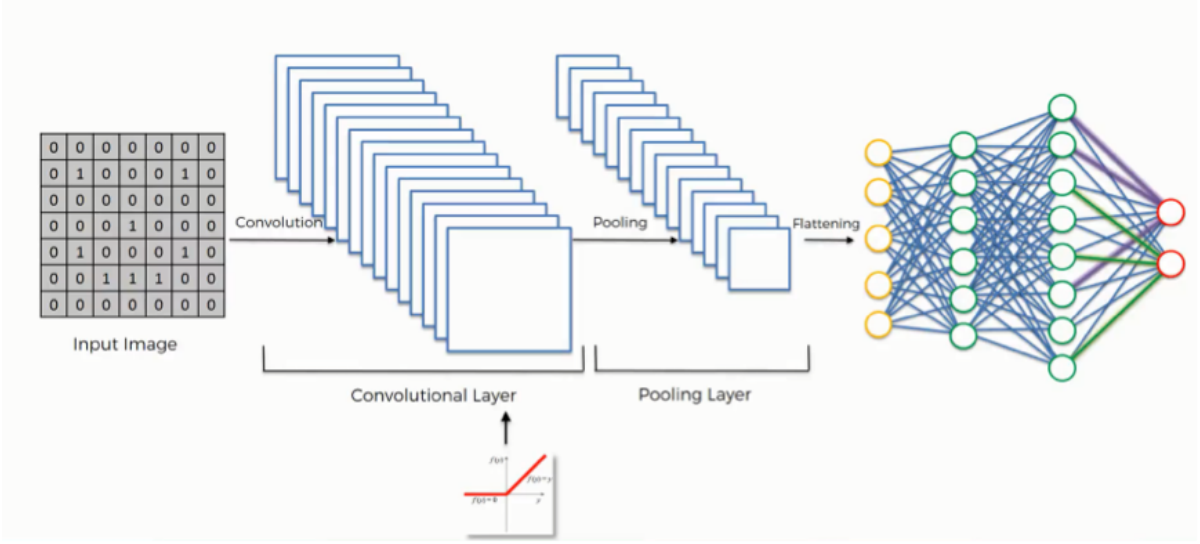
To solve the CIFAR-10 dataset, you would typically follow the following methodology:

1. Data preprocessing: The first step is to preprocess the data by converting the images into numpy array and normalizing the pixel values.
2. Model selection: The model which we choose to train the Cifar-10 dataset is a Convolution Neural Network and also a Resnet-50 model which is a pre-trained model.
3. Training: After selecting the model architecture, the model is trained on the dataset using an appropriate optimization algorithm such as stochastic gradient descent (SGD) or Adam. During training, you should monitor the training and validation loss to ensure that the model is not overfitting to the training data.
4. Hyperparameter tuning: You may need to tune the hyper parameters of the model and the optimization algorithm to achieve better performance. Hyper parameters that you cantune include learning rate, batch size, number of epochs, and regularization strength.
5. Evaluation: Once the model is trained, you need to evaluate its performance on the test dataset by computing metrics such as accuracy, precision, recall, and F1 score. You should also visualize the performance of the model using confusion matrices.
6. Fine-tuning and transfer learning: Finally, you can try fine-tuning the model on other similar datasets or using transfer learning to apply the model to other related tasks. This can help improve the performance of the model and make it more generalizable.

5.1 Convolution Neural Network

A Convolutional Neural Network (CNN) is a deep neural network sketched for processing structured arrays of data such as images. CNN are very satisfactoryat picking up on design in the input image, such as lines, gradients, circles, or even eyes and faces. This characteristic makes convolutional neural network robust for computer vision. CNN can run directly on a underdone image and do notneed any preprocessing. A convolutional neural network is a feed forward neural network. The strength of a convolutional neural network comes from a particular kind of layer called the convolutional layer. CNN contains many convolutional layers assembled on top of each other, each one competent of recognizing more sophisticated shapes. This enablesthe network to find patterns within patterns. The agenda for this sphere is to activate machines to view the world as humansdo, perceive it in a alike fashion and even use the knowledge for predicting image labels. The construction of a convolutional neural network is a multi-layered feed- forward neural network, made by assembling many unseen layers on top of each other in aparticular order. It is the sequential design that give permission to CNN to learn hierarchical attributes. In CNN, the pooling layers include

- Max Pooling choosing the most dominant feature
- Min Pooling choosing the most recessive feature
- Average Pooling choosing the general trend of features

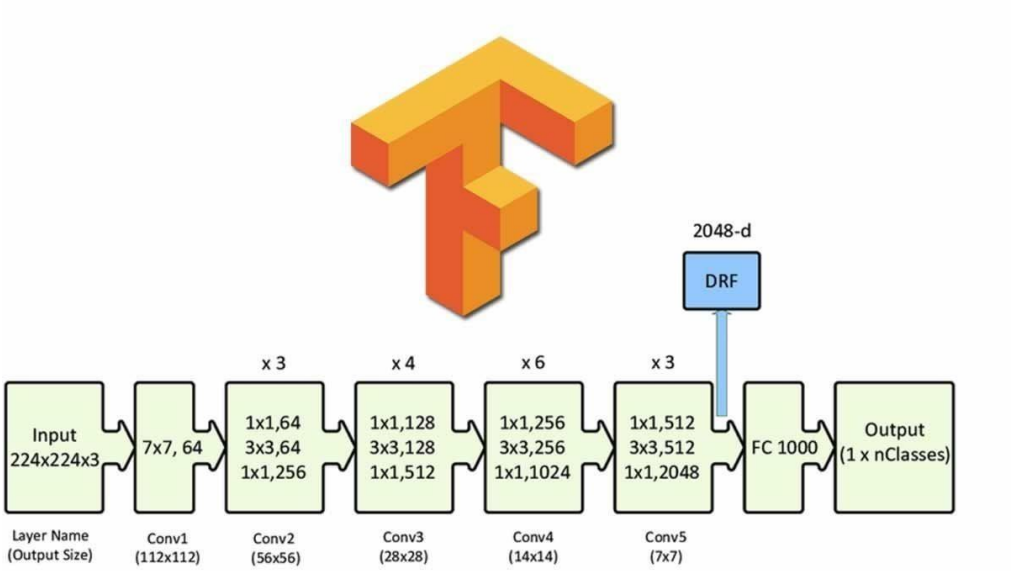


(fig 5.1: CNN)

5.2 Tensorflow

TensorFlow was originally devel- oped by researchers and engineers working on the Google Brain Team within Google'sMachine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well but mainly focused on deep neural networks. Tensorflow is the most widely used deep learning library written in c++.

TensorFlow provides a wide range of tools and APIs for building different types of models, including neural networks, decision trees, and clustering algorithms. It is widely used in various industries for applications such as image recognition, natural language processing, and data analysis. TensorFlow also has a large and active community of developers who contribute to its ongoing development and improvement.



(fig 5.2: Tensorflow)

5.3 Resnet-50 Model

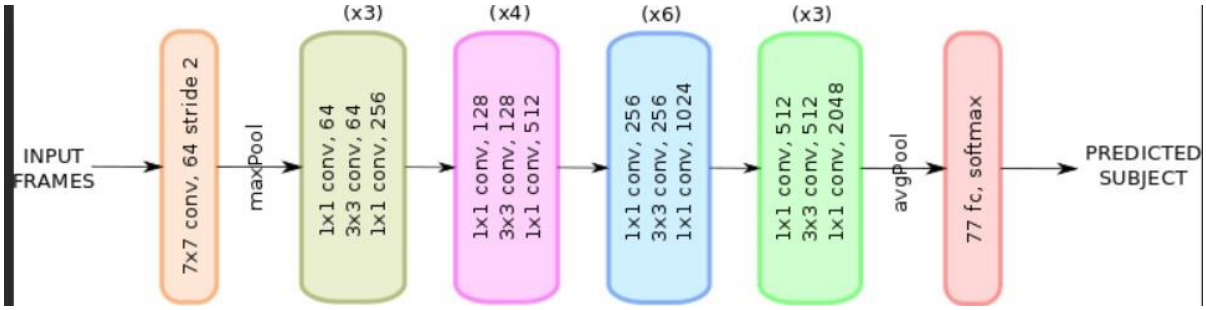
ResNet-50 is a deep convolutional neural network architecture that was introduced by Microsoft Research in 2015. It has 50 layers, hence the name ResNet-50. The architecture is a modified version of the ResNet architecture, which uses residual connections to enable deeper networks to be trained more effectively.

The ResNet-50 architecture consists of:

1. 1. Initial convolutional layer: The input image is passed through a 7x7 convolutional layer with a stride of 2, followed by max pooling with a stride of 2.

2. 2.Four stages: Each stage consists of multiple residual blocks, with each block consisting of two or three convolutional layers. The residual blocks use skip connections to add the input to the output of the block, which helps to prevent vanishing gradients and enables the network to learn more complex features.

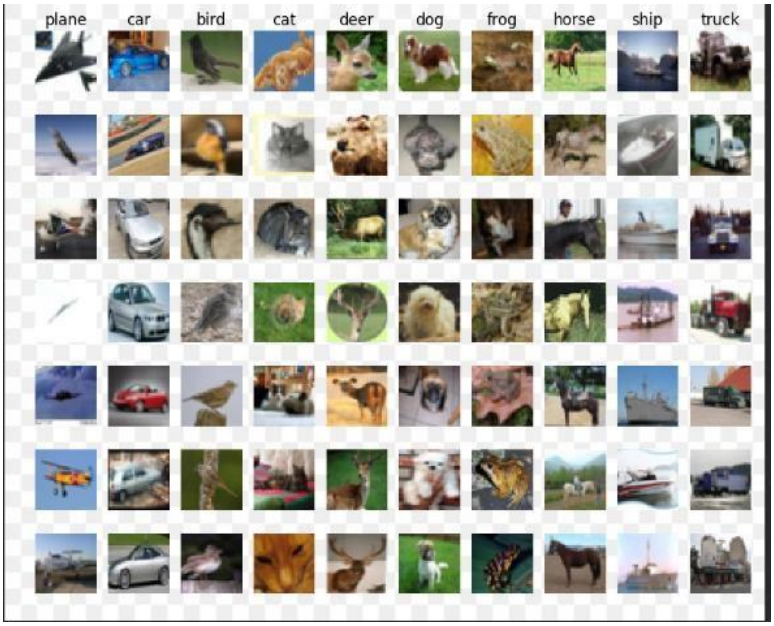
3. 3.Average pooling and fully connected layer: After the final stage, the output is passed through an average pooling layer, which computes the mean of the feature maps. The output of the average pooling layer is then passed through a fully connected layer with 1000 units, which produces the final output classification



(fig 5.3 : ResNet-50 model)

5.4 Dataset

The CIFAR-10 data consists of 60,000 32x32 color images in 10 classes, with 6000 images per class. There are 50,000 training images and 10,000 test images in the official data.



(fig 5.4: CIFAR-10 dataset)

5.5 ReLU (Rectified Linear Unit)

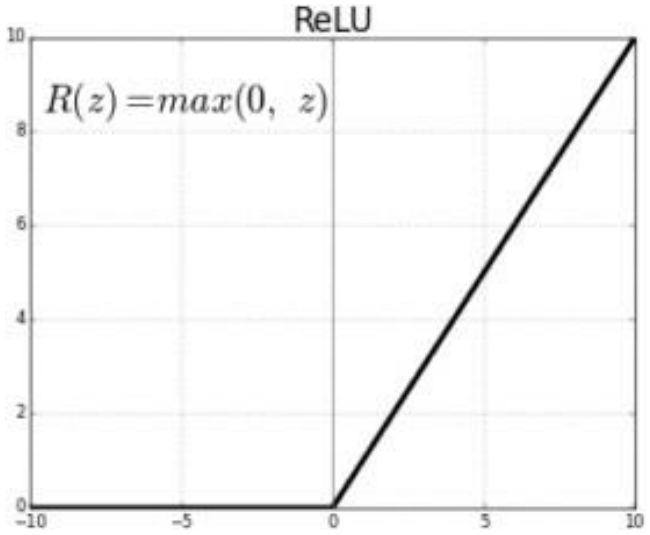
ReLU (Rectified Linear Unit) is a popular activation function used in deep learning. It is a simple yet effective function that replaces negative inputs with zero and leaves positive inputs unchanged.

Mathematically, ReLU is defined as f(x) = max(0,x), where x is the input to the function. When x is positive, the output is equal to x. When x is negative, the output is equal to zero.

ReLU is preferred over other activation functions, such as sigmoid and tanh, because it addresses the vanishing gradient problem that occurs in deep neural networks. The vanishing gradient problem occurs when the gradients become very small during backpropagation, which can slow down or even prevent the training process.

ReLU can also improve the performance of neural networks by promoting sparsity, which means that fewer neurons are activated, resulting in faster computation and reduced overfitting. However, ReLU can also suffer from a problem called "dying ReLU," where some neurons become permanently inactive due to having a negative output during training.

Overall, ReLU is a simple and effective activation function that can help improve the performance of deep neural networks.



(fig 5.5: ReLU graph)

5.6 SoftMax

Softmax is a mathematical function used to convert a vector of real numbers into a probability distribution. It is often used in machine learning for classification tasks where the output is required to be a probability distribution over a set of classes.

Given a vector of real numbers, the softmax function normalizes the values to a probability distribution by exponentiating each value and dividing by the sum of exponentiated values. The output of the softmax function is a vector where each element represents the probability of the corresponding class.

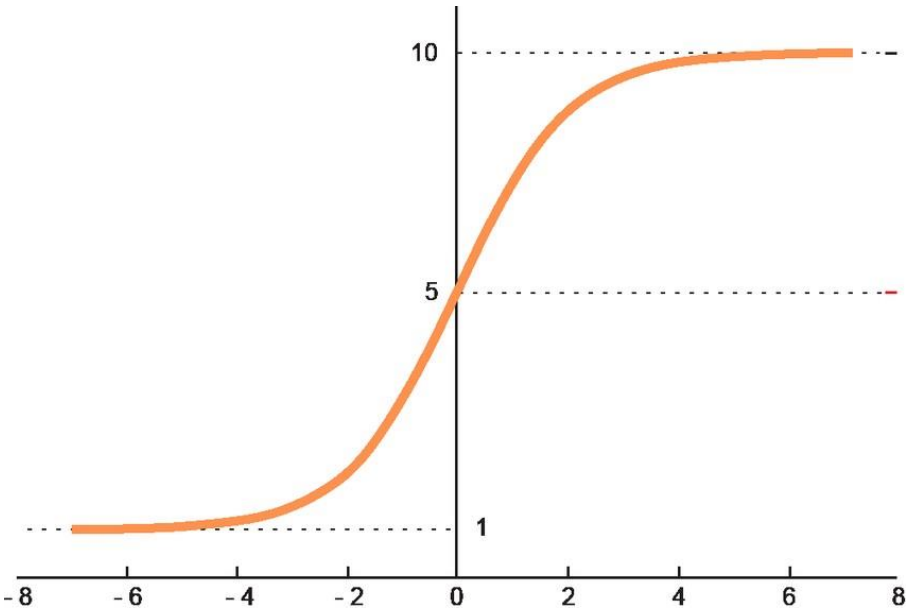
The softmax function is defined as:

softmax(x_i) = e^(x_i) / sum_j(e^(x_j))

where x is the input vector and i and j represent the indices of the elements in the vector.

Softmax has several useful properties, including that it always produces a probability distribution, meaning that the sum of the output vector is always equal to 1. Additionally, it has a natural gradient which makes it useful for training neural networks with backpropagation.

Overall, softmax is an important function for many machine learning applications, particularly in classification tasks.



(fig 5.6: SoftMax graph)

6. Experimentation

6.1 Objective

Building a CIFAR - 10 Object Recognition using ResNet50 with Transfer Learning.

6.2 Procedure

6.2.1 Creating a simple neural Network

```
# setting up the layers of Neural Network

model = keras.Sequential([

    keras.layers.Flatten(input_shape=(32,32,3)),
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dense(num_of_classes, activation='softmax')
])
```

This code defines a simple neural network model using the Keras API in TensorFlow. Here is a breakdown of what each line of code does:

- model = keras.Sequential([]): This line initializes a sequential model object in Keras. A sequential model is a linear stack of layers that are executed in order.

- keras.layers.Flatten(input_shape=(32,32,3)): This line adds a Flatten layer to the model. This layer flattens the input image from a 2D array of pixels to a 1D array. The input shape is specified as (32,32,3), indicating that the input images are 32x32 pixels with 3 color channels (RGB).

- keras.layers.Dense(64, activation='relu'): This line adds a Dense layer to the model. This layer is fully connected, meaning that each neuron in the layer is connected to every neuron in the previous layer. The layer has 64 units and uses the ReLU activation function. The ReLU activation function returns the maximum of 0 and the input value, which helps to introduce non-linearity into the model.

- keras.layers.Dense(num_of_classes, activation='softmax'): This line adds another Dense layer to the model. This layer also has a fully connected architecture, with num_of_classes units, which is the number of output classes that the model is designed to predict. The layer uses the softmax activation function, which ensures that the output values of the layer sum up to 1, and represent the probabilities of each class.

Overall, this model consists of three layers: a Flatten layer that flattens the input images, a fully connected Dense layer with ReLU activation, and another fully connected Dense layer with softmax activation that produces the final output predictions

6.2.2 Setting up the Resnet-50 Model

ResNet50

```
from tensorflow.keras import Sequential, models, layers
from tensorflow.keras.layers import Dense, Dropout, Flatten
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.models import load_model
from tensorflow.keras.models import Model
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras import optimizers
```

```
[ ] convolutional_base = ResNet50(weights='imagenet', include_top=False, input_shape=(256,256,3))
    convolutional_base.summary()
```

Building a CIFAR - 10 Object Recognition using ResNet50 with Transfer Learning. This code defines a simple neural network model using the Keras API in TensorFlow. Here is a breakdown of what each line of code does:

- model = keras.Sequential([]): This line initializes a sequential model object in Keras. A sequential model is a linear stack of layers that are executed in order.

- keras.layers.Flatten(input_shape=(32,32,3)): This line adds a Flatten layer to the model. This layer flattens the input image from a 2D array of pixels to a 1D array. The input shape is specified as (32,32,3), indicating that the input images are 32x32 pixels with 3 color channels (RGB).

- keras.layers.Dense(64, activation='relu'): This line adds a Dense layer to the model. This layer is fully connected, meaning that each neuron in the layer is connected to every neuron in the previous layer. The layer has 64 units and uses the ReLU activation function. The ReLU activation function returns the maximum of 0 and the input value, which helps to introduce non-linearity into the model.

- Keras .layers. Dense(num_of_classes, activation='softmax'): This line adds another Dense layer to the model. This layer also has a fully connected architecture, with num_of_classes units, which is the number of output classes that the model is designed to predict. The layer uses the softmax activation function, which ensures that the output values of the layer sum up to 1, and represent the probabilities of each class.

- Overall, this model consists of three layers: a Flatten layer that flattens the input images, a fully connected Dense layer with ReLU activation, and another fully connected Dense

layer. The above code defines a convolutional base using the ResNet50 architecture pre-trained on the ImageNet dataset. Here is a breakdown of what each line of code does:

- convolutional_base = ResNet50(weights='imagenet', include_top=False, input_shape=(256,256,3)): This line initializes a ResNet50 model object using Keras with pre-trained weights from the ImageNet dataset. The weights='imagenet' argument specifies that the model should be initialized with pre-trained weights from the ImageNet dataset, which includes over a million labeled images and 1000 categories. The include_top=False argument indicates that the final classification layer of the ResNet50 model should be excluded, which means that the output of the model will be the feature maps extracted from the convolutional layers of the ResNet50 model. The input_shape=(256,256,3) argument specifies the shape of the input images that will be passed to the ResNet50 model. In this case, the input images should be 256x256 pixels with 3 color channels (RGB).

- convolutional_base.summary(): This line prints a summary of the ResNet50 model architecture to the console. The summary includes information about each layer in the model, such as the layer type, output shape, and number of parameters.

Overall, this code initializes a ResNet50 model object with pre-trained weights from the ImageNet dataset, excluding the final classification layer, and prints a summary of the model architecture. The resulting model can be used as a convolutional base in transfer learning se layer with softmax activation that produces the final output predictions

6.3 Software Used

The following software were used :

Numpy

Pandas

Matplotlib

Tensorflow

Keras

6.4Challenges Faced

Limited Dataset Size: The CIFAR-10 dataset contains only 50,000 training images, which is relatively small compared to other image classification datasets. This limited dataset size can make it difficult to train a complex model like ResNet50 without overfitting.

Data Augmentation: To overcome the limitation of the dataset size, data augmentation techniques are used to generate additional training samples by performing transformations on the original

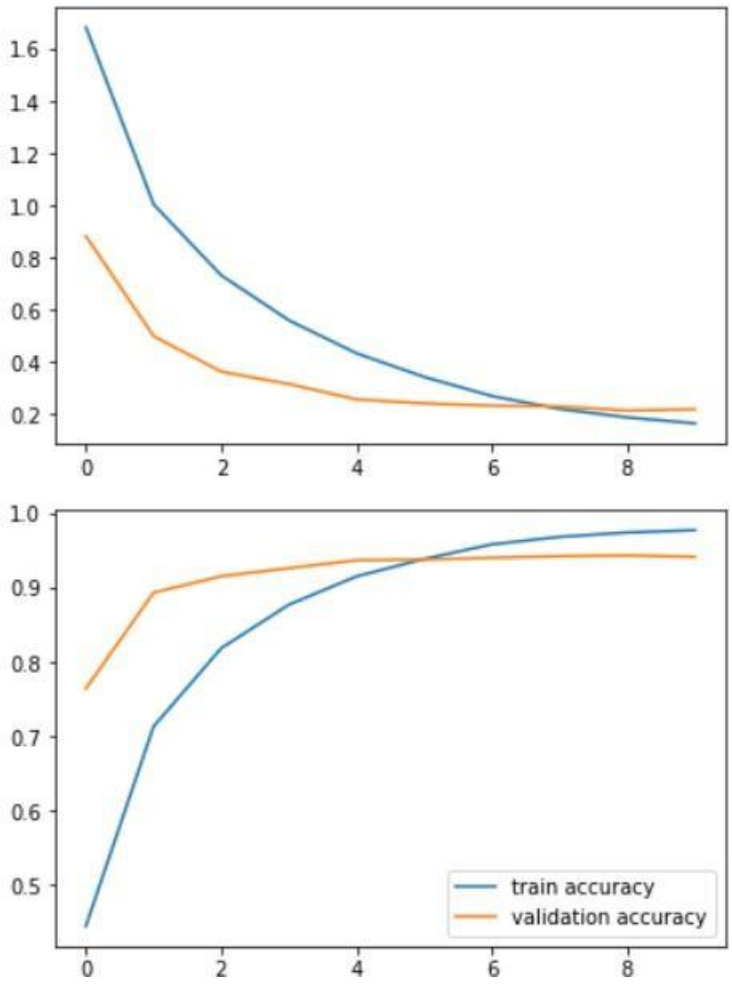
images. However, choosing the right set of augmentation techniques can be challenging, as some transformations may not be suitable for certain types of images.

Computational Resources**:** ResNet50 is a deep neural network with a large number of parameters, and training it on CIFAR-10 can be computationally intensive, requiring a significant amount of GPU memory and processing power. This can be a challenge for individuals or organizations with limited computational resources.

Fine-tuning Hyperparameters: The ResNet50 model has several hyperparameters, such as the learning rate, batch size, and weight decay, that need to be fine-tuned to achieve optimal performance on the CIFAR-10 dataset. Finding the right combination of hyperparameters can be time-consuming and requires a considerable amount of experimentation.

7. Results and discussion

7.1 Train Accuracy vs Validation Accuracy



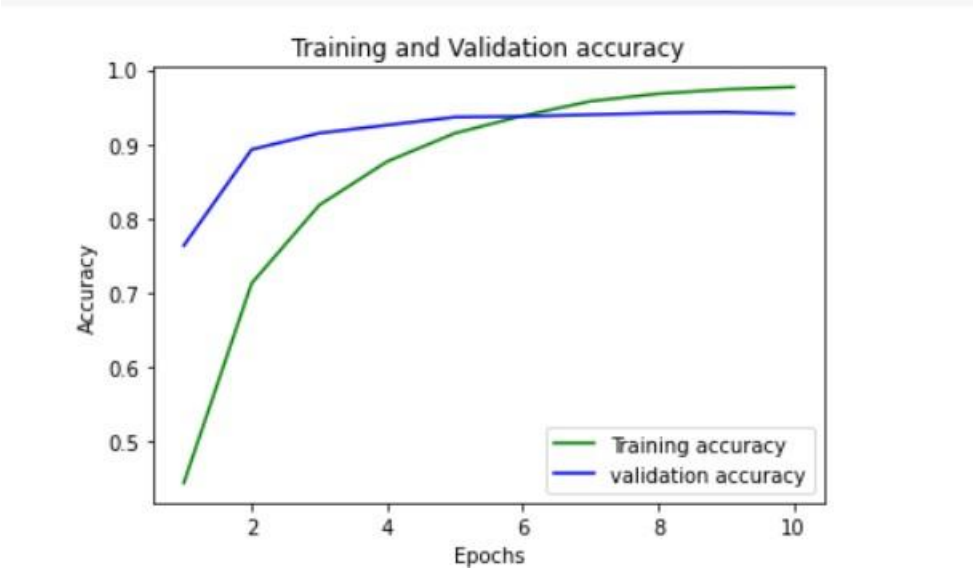
(fig 7.1: plot of train accuracy vs validation accuracy)

The training accuracy is the percentage of correctly classified examples in the training set, while the validation accuracy is the percentage of correctly classified examples in a separate validation set. The validation set is used to evaluate the performance of the model on data it has not seen during training, and to detect overfitting.

During training, it is desirable to see both the training accuracy and validation accuracy increase. However, it is not uncommon for the training accuracy to continue increasing while the validation accuracy plateaus or even decreases. This is a sign of overfitting, where the model has learned to memorize the training data instead of generalizing to new data.

Overall, the goal is to achieve a high validation accuracy, as this is a better indicator of the model's ability to generalize to new data than the training accuracy.

7.2 Accuracy vs Epoch



(fig 7.2: plot of accuracy vs epochs)

The accuracy vs epoch plot for a model trained on the CIFAR-10 dataset typically shows the change in the accuracy of the model as the number of epochs (iterations over the entire training dataset) increases during training. This plot can provide insights into how well the model is learning and whether it is overfitting.

8. Conclusion

An accuracy of 94 % was obtained on validation data. The accuracy can be further attempted to increase by adding more layer or using another Resnet model. This can be even improvised by increasing the epochs of the training phase.

Using ResNet50 on the CIFAR-10 dataset typically requires some modifications, as the network was designed for larger image sizes. One common approach is to use a modified ResNet50 architecture with smaller convolutional kernels and fewer layers.

Overall, the combination of ResNet50 and the CIFAR-10 dataset is a powerful tool for evaluating the performance of deep learning models for image classification tasks. By achieving high accuracy on this dataset, we can have confidence that the model is performing well and may be suitable for use in real-world applications.

APPENDIX A

In [1]:

```
!pip install kaggle
```

```
Looking in indexes: https://pypi.org/simple, (https://pypi.org/simple,) h
ttps://us-python.pkg.dev/colab-wheels/public/simple/ (https://us-python.p
kg.dev/colab-wheels/public/simple/)
Requirement already satisfied: kaggle in /usr/local/lib/python3.9/dist-pa
ckages (1.5.13)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.9/dist
-packages (from kaggle) (1.16.0)
Requirement already satisfied: requests in /usr/local/lib/python3.9/dist-
packages (from kaggle) (2.27.1)
Requirement already satisfied: tqdm in /usr/local/lib/python3.9/dist-pack
ages (from kaggle) (4.65.0)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.
9/dist-packages (from kaggle) (2.8.2)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.
9/dist-packages (from kaggle) (8.0.1)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.9/dist-p
ackages (from kaggle) (1.26.15)
Requirement already satisfied: certifi in /usr/local/lib/python3.9/dist-p
ackages (from kaggle) (2022.12.7)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/pyth
on3.9/dist-packages (from python-slugify->kaggle) (1.3)
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/li
b/python3.9/dist-packages (from requests->kaggle) (2.0.12)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.9/d
ist-packages (from requests->kaggle) (3.4)
```

configuring the path for kaggle.json file

In [ ]:

```
 !mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

In [ ]:

```
!kaggle competitions download -c cifar-10
```

```
Downloading cifar-10.zip to /content
100% 712M/715M [00:22<00:00, 30.1MB/s]
100% 715M/715M [00:22<00:00, 32.9MB/s]
```

linux command to print the files in current directory

In [ ]:

```
!ls
```

```
cifar-10.zip  kaggle.json  sample_data
```

In [ ]:

```
from zipfile import ZipFile
dataset = '/content/cifar-10.zip'

with ZipFile(dataset,'r') as zip:
  zip.extractall()
  print('The dataset is extracted')
```

```
The dataset is extracted
```

In [ ]:

```
!ls
```

```
cifar-10.zip  sample_data           test.7z   trainLabels.csv
kaggle.json   sampleSubmission.csv  train.7z
```

In [ ]:

```
!pip install py7zr
```

```
Looking in indexes: https://pypi.org/simple, (https://pypi.org/simple,) h
ttps://us-python.pkg.dev/colab-wheels/public/simple/ (https://us-python.p
kg.dev/colab-wheels/public/simple/)
Collecting py7zr
  Downloading py7zr-0.20.4-py3-none-any.whl (66 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 66.3/66.3 KB 6.4 MB/s eta
0:00:00
Collecting texttable
  Downloading texttable-1.6.7-py2.py3-none-any.whl (10 kB)
Collecting brotli>=1.0.9
  Downloading Brotli-1.0.9-cp39-cp39-manylinux1_x86_64.whl (357 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 357.2/357.2 KB 29.4 MB/s eta
0:00:00
Collecting pycryptodomex>=3.6.6
  Downloading pycryptodomex-3.17-cp35-abi3-manylinux_2_17_x86_64.manylinu
x2014_x86_64.whl (2.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 70.6 MB/s eta 0:
00:00
Collecting pybcj>=0.6.0
  Downloading pybcj-1.0.1-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x
86_64.whl (49 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 49.6/49.6 KB 6.4 MB/s eta
0:00:00
Collecting multivolumefile>=0.2.3
  Downloading multivolumefile-0.2.3-py3-none-any.whl (17 kB)
Collecting inflate64>=0.3.1
  Downloading inflate64-0.3.1-cp39-cp39-manylinux_2_17_x86_64.manylinux20
14_x86_64.whl (92 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 93.0/93.0 KB 2.1 MB/s eta
0:00:00
Collecting pyzstd>=0.14.4
  Downloading pyzstd-0.15.4-cp39-cp39-manylinux_2_17_x86_64.manylinux2014
_x86_64.whl (384 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 384.0/384.0 KB 43.4 MB/s eta
0:00:00
Collecting pyppmd<1.1.0,>=0.18.1
  Downloading pyppmd-1.0.0-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_
x86_64.whl (138 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 138.7/138.7 KB 21.1 MB/s eta
0:00:00
Requirement already satisfied: psutil in /usr/local/lib/python3.9/dist-pa
ckages (from py7zr) (5.9.4)
Installing collected packages: texttable, brotli, pyzstd, pyppmd, pycrypt
odomex, pybcj, multivolumefile, inflate64, py7zr
Successfully installed brotli-1.0.9 inflate64-0.3.1 multivolumefile-0.2.3
py7zr-0.20.4 pybcj-1.0.1 pycryptodomex-3.17 pyppmd-1.0.0 pyzstd-0.15.4 te
xttable-1.6.7
```

In [ ]:

```
import py7zr
archive = py7zr.SevenZipFile('/content/train.7z', mode='r')
archive.extractall()     #archive.extractall(path='/content/Training Data')
archive.close()
```

In [ ]:

```
!ls
```

```
cifar-10.zip  sample_data           test.7z  train.7z
kaggle.json   sampleSubmission.csv  train    trainLabels.csv
```

importing libraries

In [ ]:

```
import os
import numpy as np
import pandas as pd
from PIL import Image
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from sklearn.model_selection import train_test_split
```

In [ ]:

```
filenames = os.listdir('/content/train')
```

In [ ]:

```
print(filenames[0:5])
print(filenames[-5:])
```

```
['11371.png', '36310.png', '15377.png', '46595.png', '48438.png']
['21376.png', '20970.png', '8793.png', '4334.png', '8634.png']
```

labels processing

In [ ]:

```
labels_df = pd.read_csv('/content/trainLabels.csv')
```

In [ ]:

```
labels_df.shape
```

Out[14]:

```
(50000, 2)
```

In [ ]:

```
labels_df.head()
```

Out[15]:

	id	label
0	1	frog
1	2	truck
2	3	truck
3	4	deer
4	5	automobile

In [ ]:

```
labels_df[labels_df['id']== 7796]
```

Out[16]:

	id	label
7795	7796	frog

In [ ]:

```
labels_df.head(10)
```

Out[17]:

	id	label
0	1	frog
1	2	truck
2	3	truck
3	4	deer
4	5	automobile
5	6	automobile
6	7	bird
7	8	horse
8	9	ship
9	10	cat

checking for the distribution of classes

In [ ]:

```
labels_df['label'].value_counts()
```

Out[18]:

```
frog          5000
truck         5000
deer          5000
automobile    5000
bird          5000
horse         5000
ship          5000
cat           5000
dog           5000
airplane      5000
Name: label, dtype: int64
```

In [ ]:

```
labels_dictionary = {'airplane' : 0, 'automobile' : 1, 'bird' : 2,'cat':3, 'deer' : 4,'

labels = [labels_dictionary[i] for i in labels_df['label']]
```

converted the classes into numerics

In [ ]:

```
print(labels[0:5])
print(labels[-5:])
```

```
[6, 9, 9, 4, 1]
[2, 6, 9, 1, 1]
```

In [ ]:

```
#displaying sample image
import cv2
from google.colab.patches import cv2_imshow
img = cv2.imread('/content/train/45888.png')
cv2_imshow(img)
```

creating a separate list for the 'id' column

In [ ]:

```
id_list = list(labels_df['id'])
print(id_list[0:5])
print(id_list[-5:])
```

```
[1, 2, 3, 4, 5]
[49996, 49997, 49998, 49999, 50000]
```

image processing

In [ ]:

```
#converting images into numpy arrays
train_data_folder = '/content/train/'

#converting the 50k images into 50k numpy arrays into a list
data = []
for id in id_list:
  image = Image.open(train_data_folder + str(id)+ '.png')
  image = np.array(image)
  data.append(image)
```

In [ ]:

```
len(data) #checking if everything is loaded
```

Out[26]:

```
50000
```

In [ ]:

```
data[0].shape
```

Out[27]:

```
(32, 32, 3)
```

In [ ]:

```
data[0]
```

Out[28]:

```
array([[[ 59,  62,  63],
        [ 43,  46,  45],
        [ 50,  48,  43],
        ...,
        [158, 132, 108],
        [152, 125, 102],
        [148, 124, 103]],

       [[ 16,  20,  20],
        [  0,   0,   0],
        [ 18,   8,   0],
        ...,
        [123,  88,  55],
        [119,  83,  50],
        [122,  87,  57]],

       [[ 25,  24,  21],
        [ 16,   7,   0],
        [ 49,  27,   8],
        ...,
        [118,  84,  50],
        [120,  84,  50],
        [109,  73,  42]],

       ...,

       [[208, 170,  96],
        [201, 153,  34],
        [198, 161,  26],
        ...,
        [160, 133,  70],
        [ 56,  31,   7],
        [ 53,  34,  20]],

       [[180, 139,  96],
        [173, 123,  42],
        [186, 144,  30],
        ...,
        [184, 148,  94],
        [ 97,  62,  34],
        [ 83,  53,  34]],

       [[177, 144, 116],
        [168, 129,  94],
        [179, 142,  87],
        ...,
        [216, 184, 140],
        [151, 118,  84],
        [123,  92,  72]]], dtype=uint8)
```

In [ ]:

```
#converting images list and labels list to numpy arrays
X  =  np.array(data)
Y = np.array(labels)
```

In [ ]:

```
type(X)
```

Out[30]:

```
numpy.ndarray
```

In [ ]:

```
print(X.shape)
print(Y.shape)
```

```
(50000, 32, 32, 3)
(50000,)
```

In [ ]:

```
#performing train test split
X_train, X_test , Y_train, Y_test = train_test_split(X,Y,test_size = 0.2, random_state =
```

In [ ]:

```
print(X.shape , X_train.shape, X_test.shape)
```

```
(50000, 32, 32, 3) (40000, 32, 32, 3) (10000, 32, 32, 3)
```

In [ ]:

```
#scaling the data from 0-255 to 0-1

X_train_scaled = X_train/255
X_test_scaled = X_test/255
```

In [ ]:

```
X_train_scaled
```

Out[35]:

```
array([[[[0.81960784, 0.82352941, 0.79607843],
         [0.83529412, 0.83921569, 0.81960784],
         [0.85490196, 0.85882353, 0.84313725],
         ...,
         [0.49803922, 0.29019608, 0.15294118],
         [0.47843137, 0.26666667, 0.1372549 ],
         [0.45490196, 0.24705882, 0.1254902 ]],

        [[0.82352941, 0.82352941, 0.79215686],
         [0.83529412, 0.83921569, 0.81176471],
         [0.85490196, 0.8627451 , 0.83921569],
         ...,
         [0.48627451, 0.2745098 , 0.1372549 ],
         [0.4745098 , 0.2627451 , 0.12941176],
         [0.48235294, 0.27058824, 0.14117647]],

        [[0.80784314, 0.80392157, 0.76470588],
         [0.81960784, 0.81960784, 0.79215686],
```

building the NN

In [ ]:

```
import tensorflow as tf
from tensorflow import keras
```

In [ ]:

```
num_of_classes = 10

#setting up layers of neural network
model= keras.Sequential([
    keras.layers.Flatten(input_shape=(32,32,3)),
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dense(num_of_classes,activation='softmax')
])
```

In [ ]:

```
#compiling the NN
model.compile(optimizer = 'adam',
              loss='sparse_categorical_crossentropy',
              metrics=['acc'])
```

In [ ]:

```
#training the NN
model.fit(X_train_scaled ,Y_train,validation_split=0.1, epochs=10)
```

```
Epoch 1/10
1125/1125 [==============================] - 9s 4ms/step - loss: 2.0195 -
acc: 0.2480 - val_loss: 1.8979 - val_acc: 0.2982
Epoch 2/10
1125/1125 [==============================] - 3s 3ms/step - loss: 1.9009 -
acc: 0.3003 - val_loss: 1.8590 - val_acc: 0.3080
Epoch 3/10
1125/1125 [==============================] - 3s 3ms/step - loss: 1.8698 -
acc: 0.3166 - val_loss: 1.9612 - val_acc: 0.2882
Epoch 4/10
1125/1125 [==============================] - 3s 3ms/step - loss: 1.8551 -
acc: 0.3211 - val_loss: 1.8366 - val_acc: 0.3300
Epoch 5/10
1125/1125 [==============================] - 4s 3ms/step - loss: 1.8418 -
acc: 0.3249 - val_loss: 1.8220 - val_acc: 0.3370
Epoch 6/10
1125/1125 [==============================] - 3s 3ms/step - loss: 1.8281 -
acc: 0.3319 - val_loss: 1.8250 - val_acc: 0.3285
Epoch 7/10
1125/1125 [==============================] - 3s 3ms/step - loss: 1.8190 -
acc: 0.3325 - val_loss: 1.8094 - val_acc: 0.3332
Epoch 8/10
1125/1125 [==============================] - 3s 3ms/step - loss: 1.8031 -
acc: 0.3367 - val_loss: 1.8323 - val_acc: 0.3350
Epoch 9/10
1125/1125 [==============================] - 4s 4ms/step - loss: 1.7878 -
acc: 0.3482 - val_loss: 1.7671 - val_acc: 0.3568
Epoch 10/10
1125/1125 [==============================] - 4s 4ms/step - loss: 1.7790 -
acc: 0.3513 - val_loss: 1.7829 - val_acc: 0.3490
```

Out[39]:

```
<keras.callbacks.History at 0x7f2970080e50>
```

ResNet50

In [ ]:

```
#ABOUT RESNET 50- A transfer learning model , which uses a pre trained dataset to train

from tensorflow.keras import Sequential,models,layers
from tensorflow.keras.layers import Dense , Dropout, Flatten
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.models import load_model
from tensorflow.keras.models import Model
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras import optimizers
```

In [ ]:

```
convolutional_base = ResNet50(weights = 'imagenet',include_top=False,input_shape=(256,25
convolutional_base.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.
h5 (https://storage.googleapis.com/tensorflow/keras-applications/resne
t/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5)
94765736/94765736 [==============================] - 4s 0us/step
Model: "resnet50"
__________________________________________________________________________
_____________________________
 Layer (type)                   Output Shape         Param #     Conne
cted to
==========================================================================
============================
 input_1 (InputLayer)           [(None, 256, 256, 3  0           []
                                )]

 conv1_pad (ZeroPadding2D)      (None, 262, 262, 3)  0           ['inp
ut_1[0][0]']

 conv1_conv (Conv2D)            (None, 128, 128, 64  9472        ['con
```

In [ ]:

```
#adding our layers to the cnn base
model = models.Sequential()
model.add(layers.UpSampling2D((2,2))) #we upscale the image 3 times
model.add(layers.UpSampling2D((2,2)))# first time 64,64 2nd time 128,128
model.add(layers.UpSampling2D((2,2)))#3rd time 256,256
model.add(convolutional_base)
model.add(layers.Flatten())  #flatten is used to convert the matrix to vector
model.add(layers.BatchNormalization())
model.add(layers.Dense(128,activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.BatchNormalization())
model.add(layers.Dense(64,activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.BatchNormalization()) #normalizing the scaled img values after process
model.add(layers.Dense(num_of_classes, activation='softmax')) #softmax for multi class c
```

In [ ]:

```
model.compile(optimizer=optimizers.RMSprop(learning_rate=2e-5), loss= 'sparse_categorica
```

In [ ]:

```
#
history=model.fit(X_train_scaled, Y_train, validation_split=0.1, epochs=10)
```

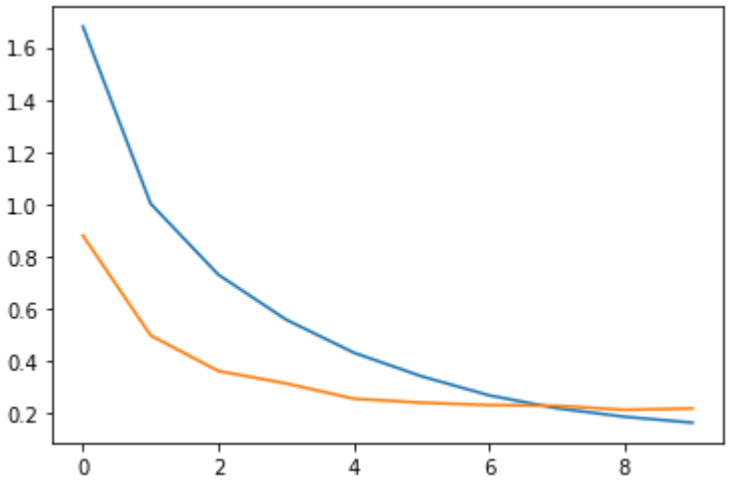
```
Epoch 1/10
1125/1125 [==============================] - 480s 396ms/step - loss: 1.67
96 - acc: 0.4449 - val_loss: 0.8804 - val_acc: 0.7638
Epoch 2/10
1125/1125 [==============================] - 446s 397ms/step - loss: 1.00
24 - acc: 0.7130 - val_loss: 0.4986 - val_acc: 0.8928
Epoch 3/10
1125/1125 [==============================] - 446s 397ms/step - loss: 0.73
09 - acc: 0.8181 - val_loss: 0.3625 - val_acc: 0.9147
Epoch 4/10
1125/1125 [==============================] - 439s 391ms/step - loss: 0.55
91 - acc: 0.8766 - val_loss: 0.3152 - val_acc: 0.9255
Epoch 5/10
1125/1125 [==============================] - 445s 396ms/step - loss: 0.43
27 - acc: 0.9146 - val_loss: 0.2568 - val_acc: 0.9362
Epoch 6/10
1125/1125 [==============================] - 439s 390ms/step - loss: 0.34
24 - acc: 0.9380 - val_loss: 0.2416 - val_acc: 0.9373
Epoch 7/10
1125/1125 [==============================] - 446s 397ms/step - loss: 0.26
94 - acc: 0.9576 - val_loss: 0.2325 - val_acc: 0.9395
Epoch 8/10
1125/1125 [==============================] - 446s 397ms/step - loss: 0.22
01 - acc: 0.9676 - val_loss: 0.2299 - val_acc: 0.9417
Epoch 9/10
1125/1125 [==============================] - 440s 391ms/step - loss: 0.18
77 - acc: 0.9735 - val_loss: 0.2141 - val_acc: 0.9427
Epoch 10/10
1125/1125 [==============================] - 440s 391ms/step - loss: 0.16
52 - acc: 0.9766 - val_loss: 0.2196 - val_acc: 0.9408
```

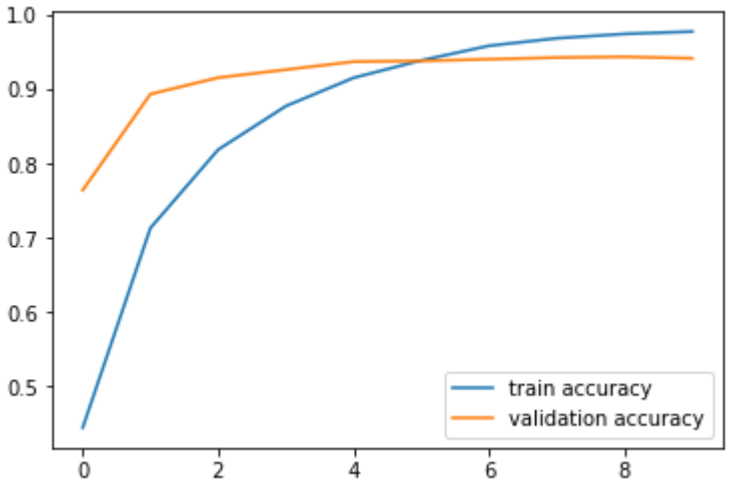
In [ ]:

```
h= history

#plotting loss value
plt.plot(h.history['loss'],label='train loss')
plt.plot(h.history['val_loss'],label = 'validation loss')
plt.show()

#plotting accuracy values
plt.plot(h.history['acc'],label='train accuracy')
plt.plot(h.history['val_acc'],label='validation accuracy')
plt.legend()
plt.show()
```





In [ ]:

```
loss_train = history.history['acc']
loss_val = history.history['val_acc']
epochs = range(1,11)
plt.plot(epochs, loss_train, 'g', label='Training accuracy')
plt.plot(epochs, loss_val, 'b', label='validation accuracy')
plt.title('Training and Validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```