Project Name: End-To-End ML with Deployment Car Brand Image Prediction – Convolution Neural Network (CNN) – ResNet50 – Classification

Table of Contents

Demo

```
# import the libraries as shown below
# state of art algorithms
from tensorflow.keras.layers import Input, Lambda, Dense, Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.applications.resnet50 import ResNet50
#from keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.resnet50 import preprocess_input
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator,load_img
from tensorflow.keras.models import Sequential
import numpy as np
from glob import glob
import matplotlib.pyplot as plt


# In[4]:


# re-size all the images to this
IMAGE_SIZE = [224, 224]

train_path = 'Datasets/train'
valid_path = 'Datasets/test'
```

```
# Import the Vgg 16 library as shown below and add preprocessing layer to the front of VGG
# Here we will be using imagenet weights

resnet = ResNet50(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)



# In[6]:


# don't train existing weights
for layer in resnet.layers:
    layer.trainable = False


# In[7]:


# useful for getting number of output classes
folders = glob('Datasets/train/*')


# In[8]:


# our layers - you can add more if you want
x = Flatten()(resnet.output)


# In[9]:


prediction = Dense(len(folders), activation='softmax')(x)
```

```
# create a model object
model = Model(inputs=resnet.input, outputs=prediction)


# In[10]:



# view the structure of the model
model.summary()


# In[11]:


# tell the model what cost and optimization method to use
model.compile(
  loss='categorical_crossentropy',
  optimizer='adam',
  metrics=['accuracy']
)
```

```
# Use the Image Data Generator to import the images from the dataset
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1./255)


# In[13]:


# Make sure you provide the same target size as initialied for the image size
training_set = train_datagen.flow_from_directory('Datasets/train',
                                                 target_size = (224, 224),
                                                 batch_size = 32,
                                                 class_mode = 'categorical')


# In[14]:


test_set = test_datagen.flow_from_directory('Datasets/test',
                                            target_size = (224, 224),
                                            batch_size = 32,
                                            class_mode = 'categorical')
```

```
# fit the model
# Run the cell. It will take some time to execute
r = model.fit_generator(
  training_set,
  validation_data=test_set,
  epochs=200,
  steps_per_epoch=len(training_set),
  validation_steps=len(test_set)
)


# In[16]:


# plot the loss
plt.plot(r.history['loss'], label='train loss')
plt.plot(r.history['val_loss'], label='val loss')
plt.legend()
plt.show()
plt.savefig('LossVal_loss')

# plot the accuracy
plt.plot(r.history['accuracy'], label='train acc')
plt.plot(r.history['val_accuracy'], label='val acc')
plt.legend()
plt.show()
plt.savefig('AccVal_acc')


# In[17]:


# save it as a h5 file
from tensorflow.keras.models import load_model
model.save('model1_resnet50.h5')
```

Live Demo Link

Deployment on Heroku:

Abstract

The purpose of this report will be to use the Car Images Data to classify brand of a given car.
This can be used to gain insight into how and why it is displayed as such brand of a car.
This can also be used as a model to gain a marketing advantage, by advertisement targeting those who are more likely to purchase luxury cars as retailers or dealers or for high-end clients.
Car Brand Image Classification is an image classification problem, where using the images, model will classify what kind of brand of car are in the images.

This is diving into Car Brand Image Classification through Machine Learning Concept.
End to End Project means that it is step by step process, starts with data collection, Resizing Data, Rescaling Data, Data Preparation which includes standardizing and transforming then selecting, training and saving DL Models, Cross-validation and Hyper-Parameter Tuning and developing web service then Deployment for end users to use it anytime and anywhere.
This repository contains the code for Car Brand Image Classification using python's various libraries.

It used numpy, matplotlib, tensorflow and keras libraries.
These libraries help to perform individually one particular functionality.
Numpy is used for working with arrays. It stands for Numerical Python.
Matplotlib is a plotting library.
TensorFlow is an open-source software library for high performance numerical computation.
Keras is built in Python which makes it way more user-friendly than TensorFlow.
These python libraries raised knowledge in discovering these libraries with practical use of it.
It leads to growth in my DL repository.
These above screenshots and video in Video_File Folder will help you to understand flow of output.

Motivation

The reason behind building this is, because till now I have worked on individual concepts so I wanted to combine all things that I have learnt till now and create an End-to-End Project that shows whole life cycle of DL Project. In addition to that, as an employee of a company, I should be able to carry out an entire process own is also essential aspect of it. Building end to end project is gave me wholesome approach to handle given data. Hence, I continue to gain knowledge while practicing the same and spread literary wings in tech-heaven.

Acknowledgement

Dataset Available: https://www.kaggle.com/ritesh2000/car-brand-images-dataset

The Data

After Downloading the dataset unzip it and place it under the Datasets Folder, all the path in the code notebook is according to my relative path so it needs to be changed accordingly.
An end-to-end application which predicts whether the car image supplied belongs to the following set of classes.

- Audi
- Lamborghini
- Mercedes

I just took baby step and start to add few more images of cars, to train my model accurately.

Modelling

- Math behind the metrics

ResNet-50 is a convolutional neural network that is 50 layers deep. You can load a pretrained version of the network trained on more than a million images from the ImageNet database. The pretrained network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. As a result, the network has learned rich feature representations for a wide range of images. The network has an image input size of 224-by-224. To retrain the network on a new classification task, follow the steps of Train Deep Learning Network to Classify New Images and load ResNet-50 instead of GoogLeNet.

What problem does ResNet solve?
Problem:
When deeper networks start converging, a degradation problem has been exposed: with the network depth increasing, accuracy gets saturated and then degrades rapidly.

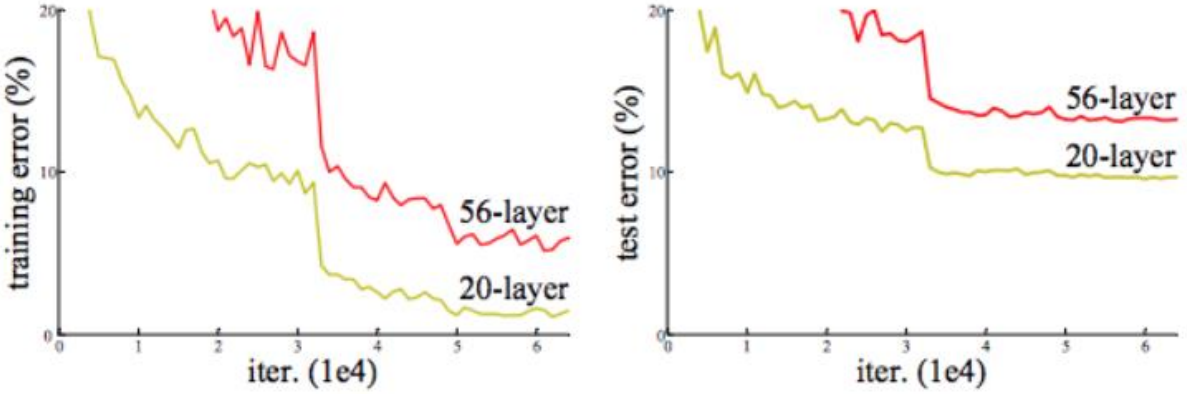
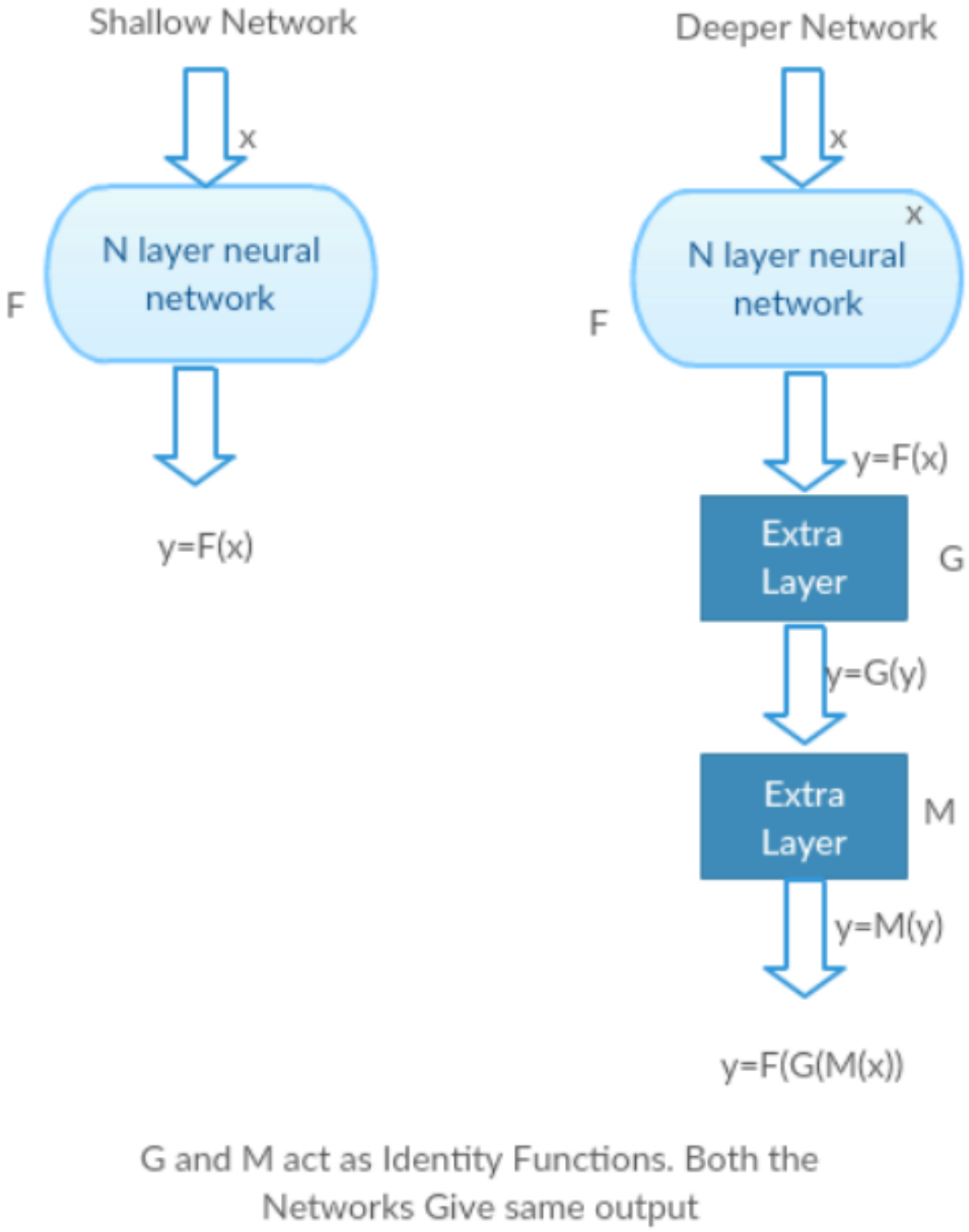


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

Seeing Degrading in Action:
Let us take a shallow network and its deeper counterpart by adding more layers to it.
Worst case scenario: Deeper model's early layers can be replaced with shallow network and the remaining layers can just act as an identity function (Input equal to output).

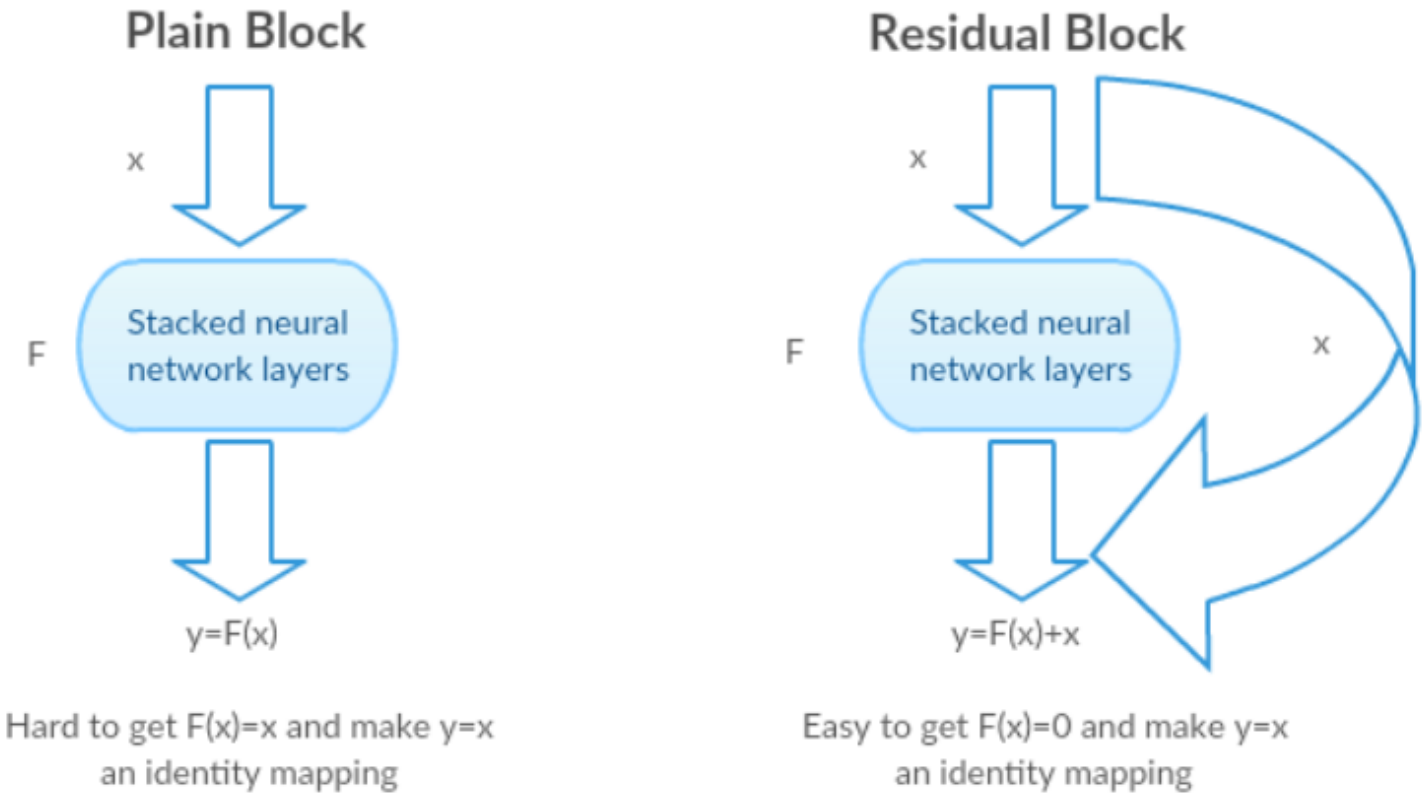


G and M act as Identity Functions. Both the Networks Give same output

Shallow network and its deeper variant both giving the same output

Rewarding scenario: In the deeper network the additional layers better approximates the mapping than its shallower counterpart and reduces the error by a significant margin.
Experiment: In the worst-case scenario, both the shallow network and deeper variant of it should give the same accuracy. In the rewarding scenario case, the deeper model should give better accuracy than its shallower counterpart. But experiments with our present solvers reveal that deeper models don't perform well. So, using deeper networks is degrading the performance of the model. This paper tries to solve this problem using Deep Residual learning framework.
How to solve?

Instead of learning a direct mapping of x ->y with a function H(x) (A few stacked non-linear layers). Let us define the residual function using F(x) = H(x) — x, which can be reframed into H(x) = F(x)+x, where F(x) and x represents the stacked non-linear layers and the identity function(input=output) respectively.
Intuition behind Residual blocks:
If the identity mapping is optimal, we can easily push the residuals to zero (F(x) = 0) than to fit an identity mapping (x, input=output) by a stack of non-linear layers. In simple language it is very easy to come up with a solution like F(x) =0 rather than F(x)=x using stack of non-linear CNN layers as function (Think about it). So, this function F(x) is what the authors called Residual function.

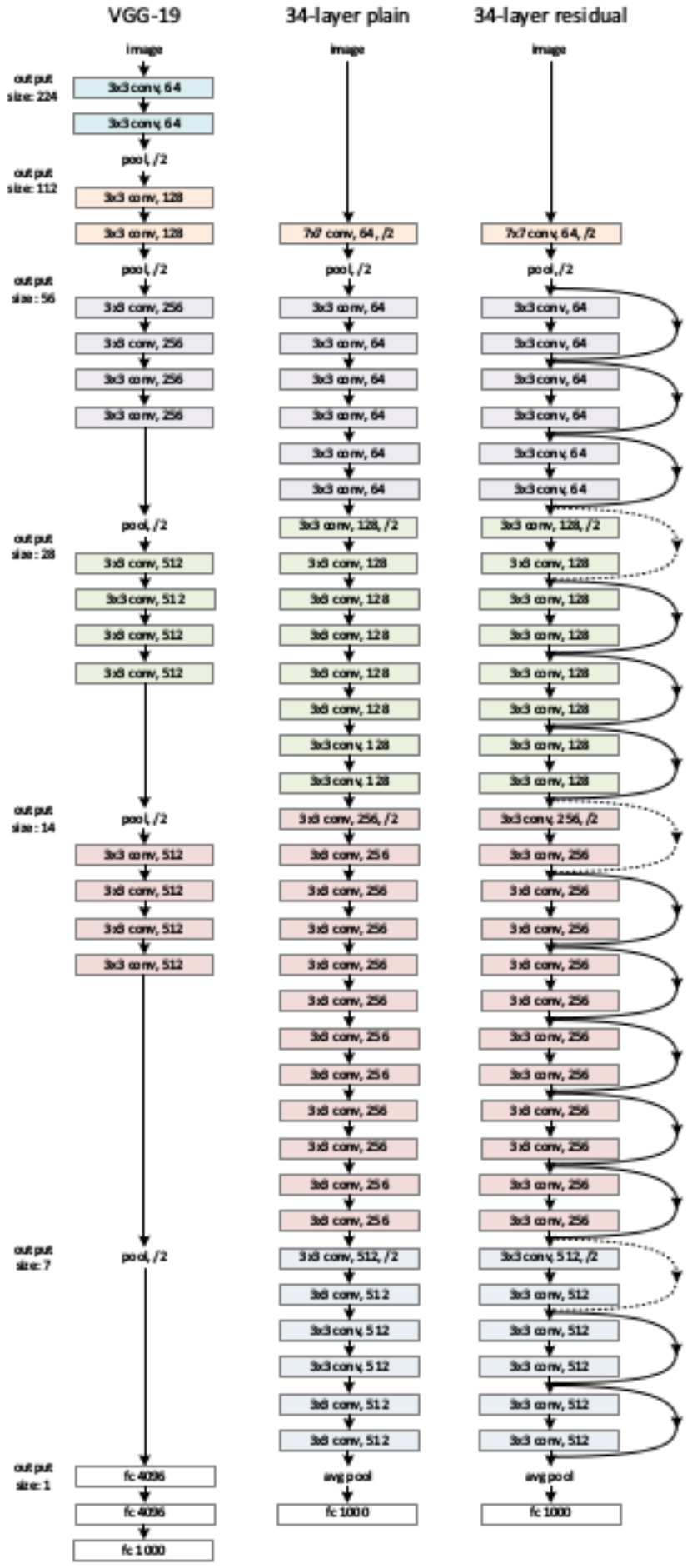


Identity mapping in Residual blocks

The Skip Connections between layers add the outputs from previous layers to the outputs of stacked layers. This results in the ability to train much deeper networks than what was previously possible. The authors of the ResNet architecture test their network with 100 and 1,000 layers on the CIFAR-10 dataset. They test on the ImageNet dataset with 152 layers, which still has less parameters than the VGG network [4], another very popular Deep CNN architecture. An ensemble of deep residual networks achieved a 3.57% error rate on ImageNet which achieved 1st place in the ILSVRC 2015 classification competition.
A similar approach to ResNets is known as “highway networks”. These networks also implement a skip connection, however, similar to an LSTM these skip connections are passed through parametric gates. These gates determine how much information passes through the skip connection. The authors note that when the gates approach being closed, the layers represent non-residual functions whereas the ResNet’s identity functions are never closed. Empirically, the authors note that the authors of the highway networks have not shown accuracy gains with networks as deep as they have shown with ResNets.

The authors made several tests to test their hypothesis. Let's look at each of them now.
Test cases:
Take a plain network (VGG kind 18-layer network) (Network-1) and a deeper variant of it (34-layer, Network-2) and add Residual layers to the Network-2 (34 layer with residual connections, Network-3).
Designing the network:

1. Use 3*3 filters mostly.
2. Down sampling with CNN layers with stride 2.
3. Global average pooling layer and a 1000-way fully-connected layer with Softmax in the end.



Plain VGG and VGG with Residual Blocks

There are two kinds of residual connections:

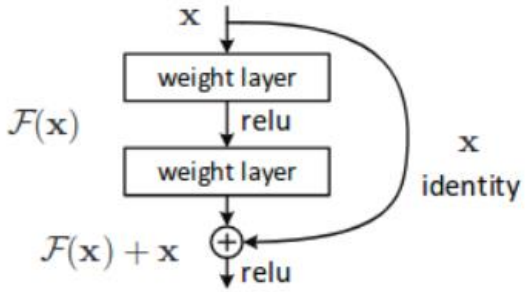


Figure 2. Residual learning: a building block.

Residual block

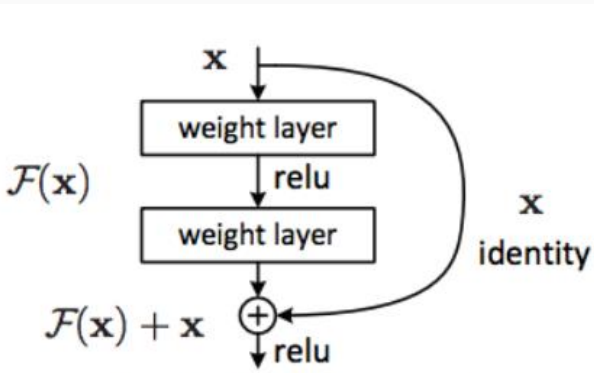


Figure 2. Residual learning: a building block.

For developers looking to quickly implement this and test it out, the most important modification to understand is the 'Skip Connection', identity mapping. This identity mapping does not have any parameters and is just there to add the output from the previous layer to the layer ahead. However, sometimes x and F(x) will not have the same dimension. Recall that a convolution operation typically shrinks the spatial resolution of an image, for example, a 3x3 convolution on a 32 x 32 image results in a 30 x 30 image. The identity mapping is multiplied by a linear projection W to expand the channels of shortcut to match the residual. This allows for the input x and F(x) to be combined as input to the next layer.

1. The identity shortcuts (x) can be directly used when the input and output are of the same dimensions.

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}. \qquad (1)$$

Residual block function when input and output dimensions are same

2. When the dimensions change, A) The shortcut still performs identity mapping, with extra zero entries padded with the increased dimension. B) The projection shortcut is used to match the dimension (done by 1*1 conv) using the following formula

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + W_s\mathbf{x}. \qquad (2)$$

Residual block function when the input and output dimensions are not same.

The first case adds no extra parameters, the second one adds in the form of W_{s}

Results:
Even though the 18-layer network is just the subspace in 34-layer network, it still performs better. ResNet outperforms by a significant margin in case the network is deeper.

	plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	**25.03**

Table 2. Top-1 error (%, 10-crop testing) on ImageNet validation. Here the ResNets have no extra parameter compared to their plain counterparts. Fig. 4 shows the training procedures.

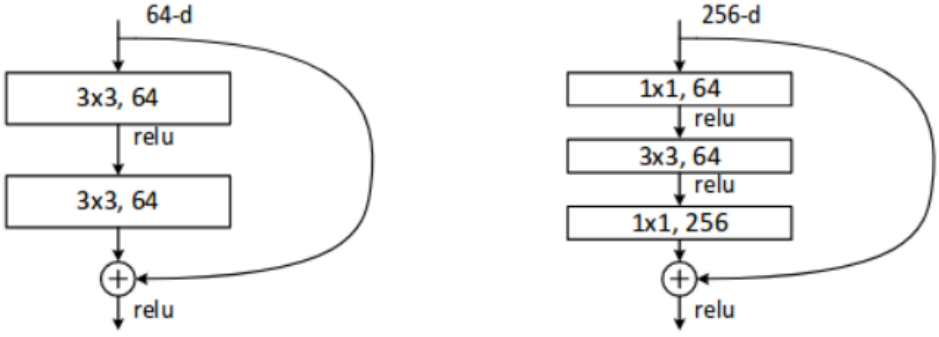
ResNet Model comparison with their counter plain nets

Deeper Studies:
The following networks are studied.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

ResNet Architectures

Each ResNet block is either 2 layer deep (Used in small networks like ResNet 18, 34) or 3 layer deep(ResNet 50, 101, 152).



ResNet 2 layer and 3 layer Block

The Bottleneck class implements a 3-layer block and Basic block implements a 2-layer block. It also has implementations of all ResNet Architectures with pretrained weights trained on ImageNet.
Observations:

1. ResNet Network Converges faster compared to plain counter part of it.
2. Identity vs Projection shortcuts. Very small incremental gains using projection shortcuts (Equation-2) in all the layers. So, all ResNet blocks use only Identity shortcuts with Projections shortcuts used only when the dimensions changes.
3. ResNet-34 achieved a top-5 validation error of 5.71% better than BN-inception and VGG. ResNet-152 achieves a top-5 validation error of 4.49%. An ensemble of 6 models with different depths achieves a top-5 validation error of 3.57%. Winning the 1st place in ILSVRC-2015.

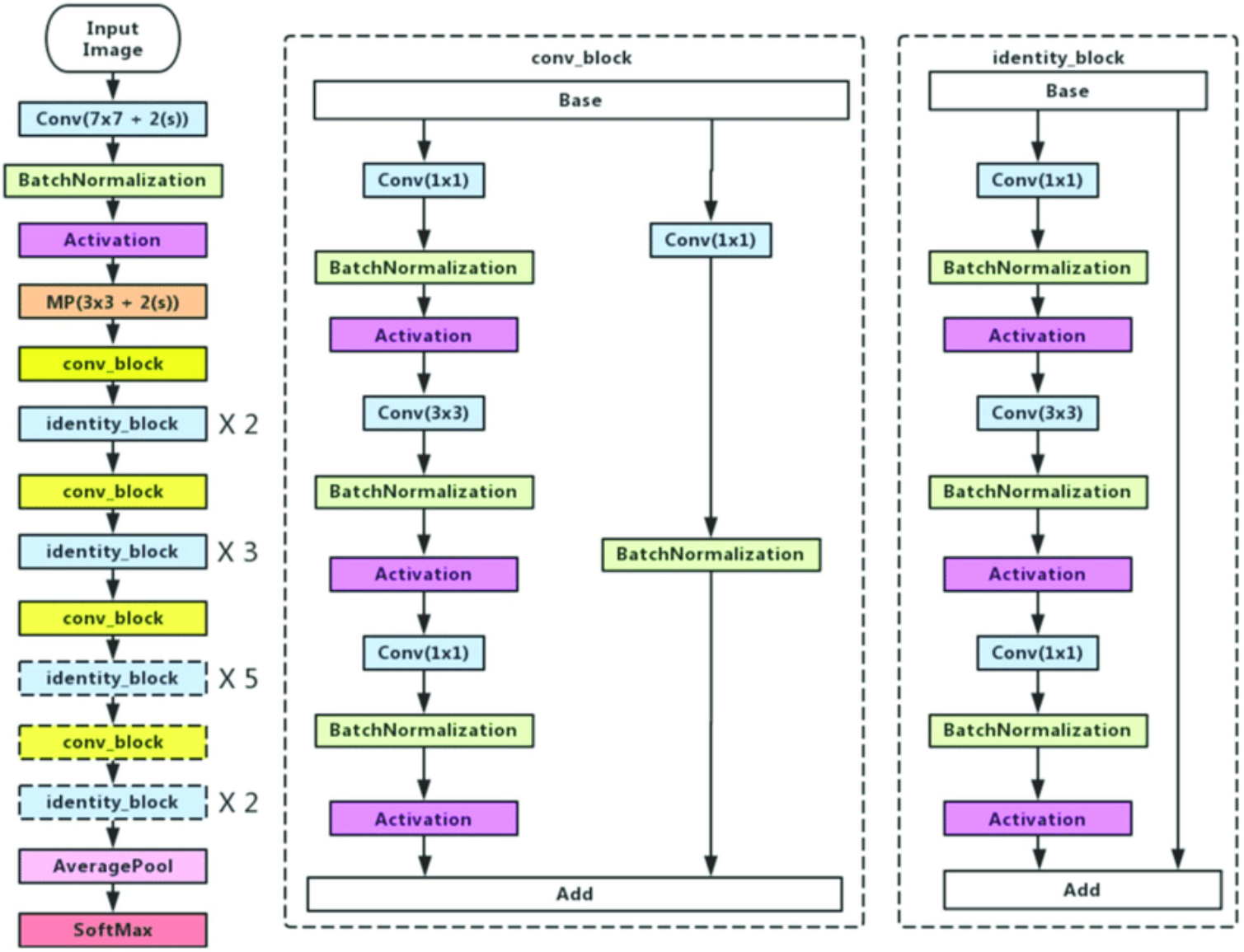
method	top-1 err.	top-5 err.
VGG [41] (ILSVRC'14)	-	8.43†
GoogLeNet [44] (ILSVRC'14)	-	7.89
VGG [41] (v5)	24.4	7.1
PReLU-net [13]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

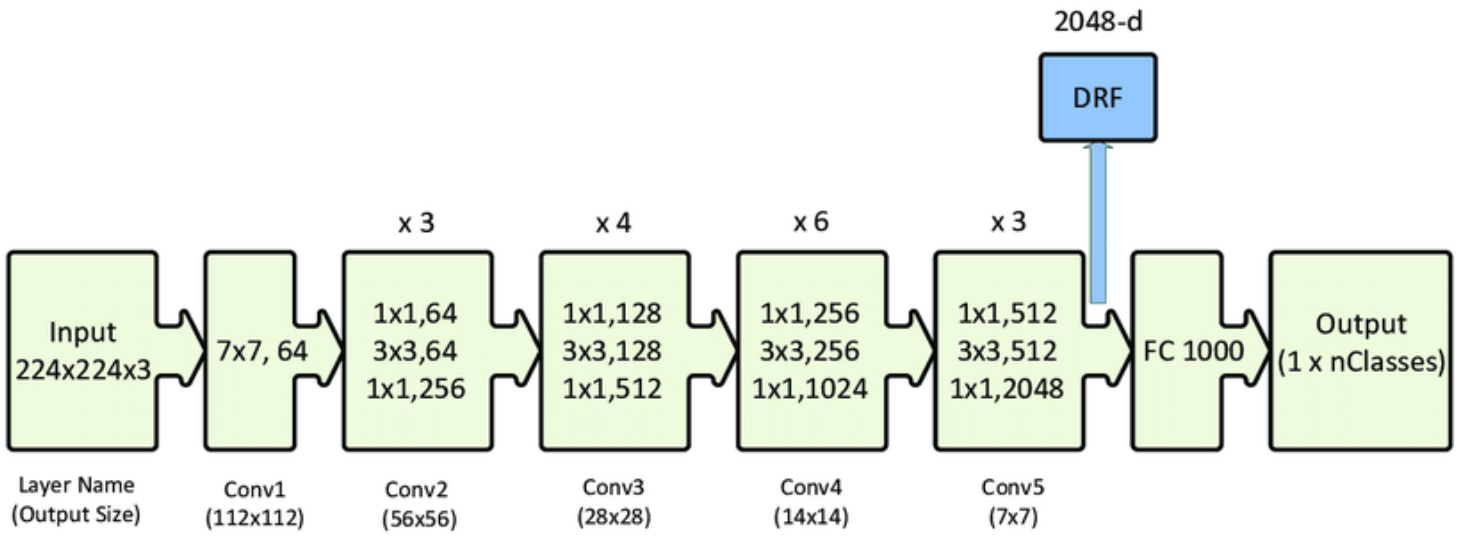
Table 4. Error rates (%) of **single-model** results on the ImageNet validation set (except † reported on the test set).

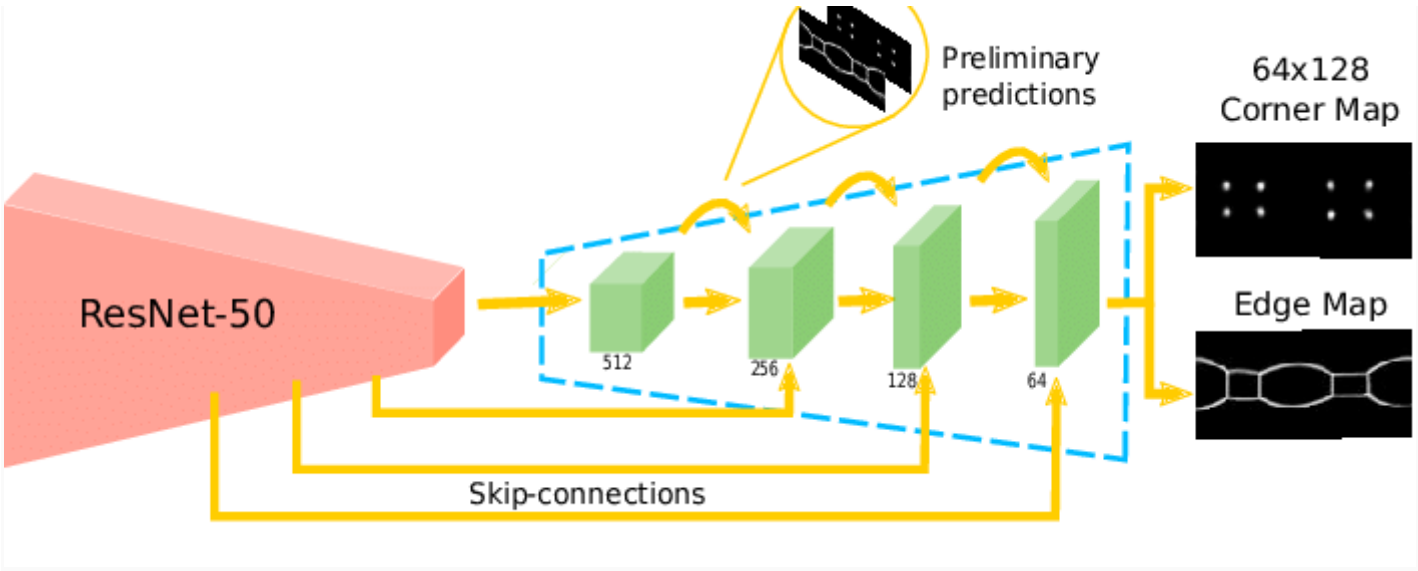
ResNet ImageNet Results-2015

Reference: https://www.tensorflow.org/api_docs/python/tf/keras/applications/ResNet50

- Model Architecture Process Through Visualization







- Quick Notes

Step 1: Imported required libraries.
Step 2: Loaded and Read the data.
Step 3: Resized images.
Step 4: Added a layer to weights.
Step 5: Used default weights which are used by Imagenet.
Step 6: Calculated number of output classes.
Step 7: Set the flatten layer.
Step 8: Created model object.
Step 9: Viewed structure of the model.
Step 10: Directed model what cost and optimization method to use.
Step 11: Used the Image Data Generator to import the images from the dataset.
Step 12: Provided the same target size as initialized for the image size.
Step 13: Fitted the model.
Step 14: Plotted loss and accuracy.
Step 15: Saved the generated model.
Step 16: Performed prediction on test data.
Step 17: Loaded the model that was saved.
Step 18: Predicted image from new dataset.
Step 19: Created Web App.

- The Model Analysis

Imported required libraries - When we import modules, we're able to call functions that are not built into Python. Some modules are installed as part of Python, and some we will install through pip. Making use of modules allows us to make our programs more robust and powerful as we're leveraging existing code.

Loaded and Read Data - Images are typically in PNG or JPEG format. You can load image data from folders in .jpg format.

Resized Images - Resizing images is a critical pre-processing step in computer vision. Principally, our machine learning models train faster on smaller images. An input image that is twice as large requires our network to learn from four times as many pixels and that time adds up. When you resize an image and do not resample it, you change the image's size without changing the

amount of data in that image. The only two values you can change are the physical size (Width and Height in Document Size) or the resolution (pixels/inch).

Added a layer to weights – ResNet50 is a model for transfer learning. Keras provides access to a number of top-performing pre-trained models that were developed for image recognition tasks. When loading a given model, the "*include_top*" argument can be set to *False*, in which case the fully-connected output layers of the model used to make predictions is not loaded, allowing a new output layer to be added and trained. And specify the preferred shape of the images in our new dataset as [3] means making images into 3 channels. Transfer learning is flexible, allowing the use of pre-trained models directly as feature extraction preprocessing and integrated into entirely new models. We load the imagenet weights for the networks.
Used default weights which are used by Imagenet - A trained model has two parts – Model Architecture and Model Weights. First, train only the top layers (which were randomly initialized) i.e. freeze all convolutional ResNet50 layers.
Calculated number of output classes – It displays number of classes as in this case, there are three classes namely Audi, Lamborghini and Mercedes.
Set the flatten layer - A flatten layer collapses the spatial dimensions of the input into the channel dimension. For example, if the input to the layer is an H-by-W-by-C-by-N-by-S array (sequences of images), then the flattened output is an (H*W*C)-by-N-by-S array. This layer supports sequence input only. Flatten is the function that converts the pooled feature map to a single column that is passed to the fully connected layer. Dense adds the fully connected layer to the neural network. Fully Connected layers in a neural-networks are those layers where all the inputs from one layer are connected to every activation unit of the next layer. In most popular machine learning models, the last few layers are full connected layers which compiles the data extracted by previous layers to form the final output.

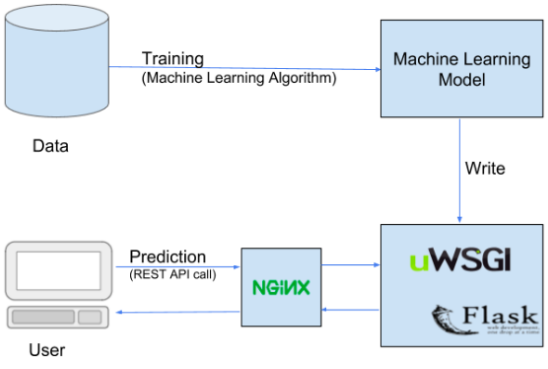
Created model object - The pre-trained model may be used as a standalone program to extract features from new photographs. We will load the model with the classifier output part of the model.
Viewed structure of the model – Seeing structure of model that we took and total number of parameters it has.

Directed model what cost and optimization method to use – Compiling the model here using compile method. Compile defines the loss function, the optimizer and the metrics. You need a compiled model to train because training uses the loss function and the optimizer.
The compile method requires several parameters. The loss parameter is specified to have type 'categorical_crossentropy'. The metrics parameter is set to 'accuracy' and finally we use the adam optimizer for training the network. It indicates that model is now ready to use.
Used the Image Data Generator to import the images from the dataset - The Keras deep learning neural network library provides the capability to fit models using image data augmentation via the ImageDataGenerator class. Image data augmentation is used to expand the training dataset in order to improve the performance and ability of the model to generalize.
ImageDataGenerator accepts the original data, randomly transforms it, and returns only the new, transformed data.
Provided the same target size as initialized for the image size - Folder should also contain a single folder inside which all the test images are present. "unlabeled" class, this is there because the flow_from_directory() expects at least one directory under the given directory path. The folder

names for the classes are important, name(or rename) them with respective label names so that it would be easy for you later.
Most used attributes along with the flow_from_directory() method.

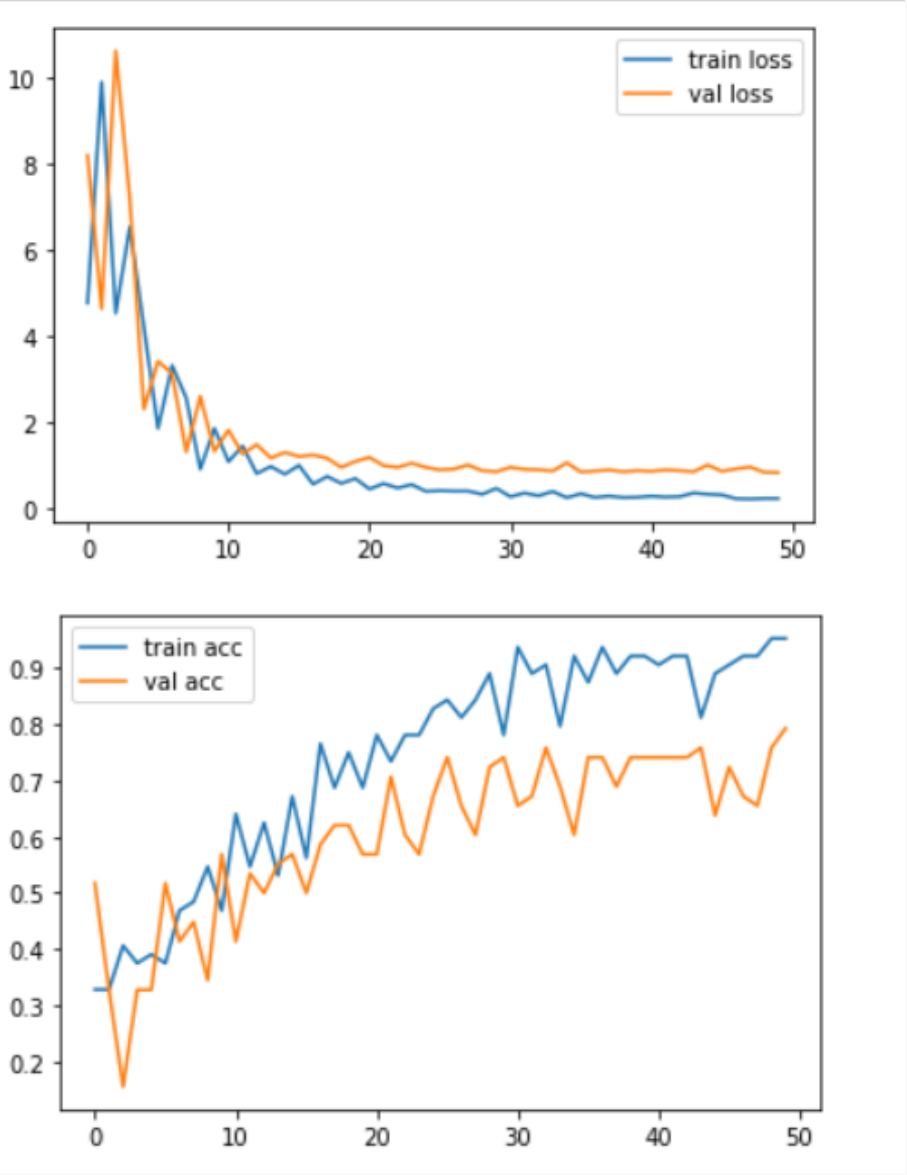
1. The directory must be set to path where your 'n' classes of folders are present.
2. The target_size is the size of your input images, every image will be resized to this size.
3. batch_size indicates number of images to be yielded from the generator per batch.
4. class_mode, it set "binary" if you have only two classes to predict, if not set to "categorical" in case if you are developing an Autoencoder system, both input and the output would probably be the same image, for this case set to "input".

Fitted the model - The .fit_generator() function accepts the batch of data, performs backpropagation, and updates the weights in our model. This process is repeated until we have reached the desired number of epochs. Supply a steps_per_epoch parameter when calling .fit_generator(). Keep in mind that a Keras data generator is meant to loop infinitely, it should *never* return or exit. Since the function is intended to loop infinitely, Keras has no ability to determine when *one epoch starts* and a new epoch begins. Therefore, we compute the steps_per_epoch value as the total number of training data points divided by the batch size. Once Keras hits this step count it knows that it's a new epoch.
Plotted loss and accuracy - Visualizing the training loss vs. validation loss or training accuracy vs. validation accuracy over a number of epochs is a good way to determine if the model has been sufficiently trained. This is important so that the model is not undertrained and not overtrained such that it starts memorizing the training data which will, in turn, reduce its ability to predict accurately.
Saved the generated model – Saving the model to re-use directly wherever needed instead of going through whole process again of training again.
Performed prediction on test data – It shows likelihood of a particular outcome. To start making predictions, you'll use the testing dataset in the model that you've created. Keras enables you to make predictions by using the .predict() function. Since you've already trained the classifier with the training set, this code will use the learning from the training process to make predictions on the test set. This will give you the probabilities of a brand of car identification.
Loaded the model that was saved – Calling the model that was previously saved, this basically saves our time to train again at that point of time.
Predicted image from new dataset – Checking the performance.
Created Web App – This is to make useful for end-user.
Challenge that I faced in this project is, model was not able to predict the image accurately so, I added more images manually to dataset so that model could learn better and also, increased number of epochs to train it properly. This, helped me and later my model could predict with better accuracy.
Reference: https://www.kaggle.com/dskagglemt/car-image-classification-using-resnet50
https://www.datacamp.com/community/tutorials/cnn-tensorflow-python
https://www.digitalocean.com/community/tutorials/how-to-build-a-deep-learning-model-to-predict-employee-retention-using-keras-and-tensorflow



Checking the Model Visualization

- Basic Model Evaluation Graphs



Creation of App

Here, I am creating Flask App. Loading the model that we saved then converting an image to array and normalizing it. That is, pre-processing the supplied image.
Get input image from client then predict class and render respective .html page for solution.
You can create it with Streamlit also.

Technical Aspect

Numpy used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. It contains a multi-dimensional array and matrix data structures. It can be utilised to perform a number of mathematical operations on arrays.
Matplotlib is used for EDA. Visualization of graphs helps to understand data in better way than numbers in table format. Matplotlib is mainly deployed for basic plotting. It consists of bars, pies, lines, scatter plots and so on. Inline command display visualization inline within frontends like in Jupyter Notebook, directly below the code cell that produced it.
Tensorflow provides flexible architecture allows easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices.

Keras allows users to productize deep models on smartphones (iOS and Android), on the web, or on the Java Virtual Machine. Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow.
Flask is a web framework. This means flask provides you with tools, libraries and technologies that allow you to build a web application. This web application can be some web pages, a blog, a wiki or go as big as a web-based calendar application or a commercial website.

Installation

Using intel core i5 9th generation with NVIDIA GFORECE GTX1650.
Windows 10 Environment Used.
Already Installed Anaconda Navigator for Python 3.x
The Code is written in Python 3.8.
If you don't have Python installed then please install Anaconda Navigator from its official site.
If you are using a lower version of Python you can upgrade using the pip package, ensuring you have the latest version of pip, *python* -m pip install --*upgrade pip and press Enter*.

Run/How to Use/Steps

Keep your internet connection on while running or accessing files and throughout too.
Follow this when you want to perform from scratch.
Open Anaconda Prompt, Perform the following steps:
cd <PATH>
pip install tensorflow==2.2.0
pip install matplotlib
pip install numpy==1.18.4 or 1.19.3
pip install flask==1.1.2
Note: If it shows error as 'No Module Found' , then install relevant module.
You can also create requirement.txt file as, pip freeze > requirements.txt
Create Virtual Environment:
conda create -n ee2 python=3.6
y
conda activate ee2
cd <PATH-TO-FOLDER>
run .py or .ipynb files.
Paste URL to browser to check whether working locally or not.

Follow this when you want to just perform on local machine.
Download ZIP File.
Right-Click on ZIP file in download section and select Extract file option, which will unzip file.
Move unzip folder to desired folder/location be it D drive or desktop etc.
Open Anaconda Prompt, write cd <PATH> and press Enter.
eg: cd C:\Users\Monica\Desktop\Projects\Python Projects 1\
23)End_To_End_Projects\Project_2_DL_FileUse_EndToEnd_CarBrandImageClassification\
Project_DL_CarImageClassification

conda create -n ee2 python=3.6
y
conda activate ee2
In Anconda Prompt, pip install -r requirements.txt to install all packages.
In Anconda Prompt, write python app.py and press Enter.
Paste URL 'http://127.0.0.1:5000/' to browser to check whether working locally or not.
Please be careful with spellings or numbers while typing filename and easier is just copy filename and then run it to avoid any silly errors.
Note: cd <PATH>
[Go to Folder where file is. Select the path from top and right-click and select copy option and paste it next to cd one space <path> and press enter, then you can access all files of that folder]
[cd means change directory]

Directory Tree/Structure of Project

Folder: 23)End_To_End_Projects > Project_2_DL_FileUse_EndToEnd_CarImageClassification> Project_DL_CarImageClassification

Model_Building.py

'Datasets' Folder

'uploads' Folder

Procfile

app.py

model1_resnet50.h5

requirements.txt

Folder: 23)End_To_End_Projects > Project_2_DL_FileUse_EndToEnd_CarImageClassification > Project_DL_CarImageClassification > templates

index.html

base.html

Folder: 23)End_To_End_Projects > Project_2_DL_FileUse_EndToEnd_CarImageClassification > Project_DL_CarImageClassification > static > css

main.css

Folder: 23)End_To_End_Projects > Project_2_DL_FileUse_EndToEnd_CarImageClassification > Project_DL_CarImageClassification > static > js

main.js

```
C:.
└───Project_DL_CarImageClassification
    ├───Datasets
    │   ├───Test
    │   │   ├───audi
    │   │   ├───lamborghini
    │   │   └───mercedes
    │   └───Train
    │       ├───audi
    │       ├───lamborghini
    │       └───mercedes
    ├───Images_screenshot
    ├───static
    │   ├───css
    │   │   └───main.css
    │   └───js
    │       └───main.js
    ├───templates
    │   ├───base.html
    │   └───index.html
    ├───uploads
    ├───app.py
    ├───Model_Building.py
    ├───model1_resnet50.h5
    ├───Procfile
    └───requirements.txt
```

To Do/Future Scope

Can Deploy on AWS.

Technologies Used/System Requirements/Tech Stack

Download the Material

You can Download Dataset from here: https://github.com/monicadesAI-tech/Project_66/tree/main/Datasets
You can Download Entire Project from here: https://github.com/monicadesAI-tech/Project_66
You can Download Model Building from here: https://github.com/monicadesAI-tech/Project_66/blob/main/Model_Building.py
Download Saved Model from here:https://github.com/monicadesAI-tech/Project_66/blob/main/model1_resnet50.h5
You can Download Web App_File from here: https://github.com/monicadesAI-tech/Project_66/blob/main/app.py
Download dependencies: https://github.com/monicadesAI-tech/Project_66/blob/main/requirements.txt
You can see Detailed Project at Website here: https://github.com/monicadesAI-tech.github.io/project/carbrand.html
Download sample images to test app: https://github.com/monicadesAI-tech/Project_66/tree/main/sample_imgs_to_test_app

Conclusion

- Modelling

Using more number images and increasing epochs will help. Along with that, training on GPU will add little significance to it.

- Analysis

Accuracy achieved here is approximately 80%.

Credits

Krish Naik Channel

Paper Citation

https://arxiv.org/abs/1512.03385

https://arxiv.org/pdf/1904.07568.pdf

https://keras.io/api/applications/