Cat and Dog Classification Using Transfer Learning

PROJECT REPORT

Last updated: 2012-09-05 at 23:42:01

MUHAMMAD SAAD FARRUKH

CMS ID: 373188

SEECS, NUST

Table of Contents

UNIT: 01 INTRODUCTION

In this document, we are going to build a PyTorch Image Classifier using **Transfer Learning,** we will test different models, pre-trained on ImageNet.

The **ImageNet Project** is a large visual database designed for use in visual object recognition software research. More than 14 million images have been hand-annotated by the project to indicate what objects are pictured and in at least one million of the images, bounding boxes are also provided. This dataset is organized according to the WorldNet Hierarchy.

After applying feature extraction to different models, we will compare their losses and inference time to select a model which will be used to classify Cats and Dogs images.

We will also spend time on test model based upon **Quantization.** Quantization is the process of mapping continuous infinite values to a smaller set of discrete finite values; resulting in greater inference time with the main objective of hardware implementation.

Also, we are also going to create an **API** of our best performing model for providing easy to use interface for user. The Machine Learning API describes the machine learning data types, and includes the API for creating, deleting, or updating a transform, or starting a machine learning task run.

After testing various models, we found **MobileNet v2** to be most effective with respect to both accuracy and inference time. Let’s see how!

UNIT: 02 THEORETICAL UNDERSTANDING

2.1 Dataset

2.1.1 Original KAGGLE Dataset

Kaggle is an online community platform for data scientists and machine learning enthusiasts. The dogs and cats dataset, The ASIRRA (animal species image recognition for restricting access) was first introduced for a Kaggle competition in 2013.

Original dataset has 12500 images of dogs and 12500 images of cats, in 25000 images in total. That's a huge amount to train the model To access the dataset, you will need to create a Kaggle account and to log in, and download dataset from following link

https://www.kaggle.com/c/dogs-vs-cats/data (853.96 MB)

Or run the following command,

```
kaggle competitions download -c dogs-vs-cats
```

2.1.2 Given Dataset

In our case, we just only use **600** images for training, **300** images for validation, and **300** images for test, in .npy (numpy arrays) formatted separated for images and labels.

The training, validation and testing datasets required in this case has been downloaded from the following links:

- **images_train:** https://drive.google.com/open?id=1r0G39tYaSe7fObGIw6vlxUhv-HkENu4W
- **images_valid:** https://drive.google.com/open?id=1_eMoeMi4KjTbcZpBr1Elz8gNU5SNIyiD
- **images_test:** https://drive.google.com/open?id=1yLGqMixFsqzVznWU8rWrJ9ZHjVDIDTxG
- **labels_train:** https://drive.google.com/open?id=1BEIuwRmS3Md1FgtdW6_45xiH-ysOWjV4
- **labels_valid:** https://drive.google.com/open?id=19SS_jVTPqISO36R6Gia58MSAxAOT4-a8
- **labels_test:** https://drive.google.com/open?id=1KGBMlxxA7b6m2rOIIM3DH2462ZUsP96z

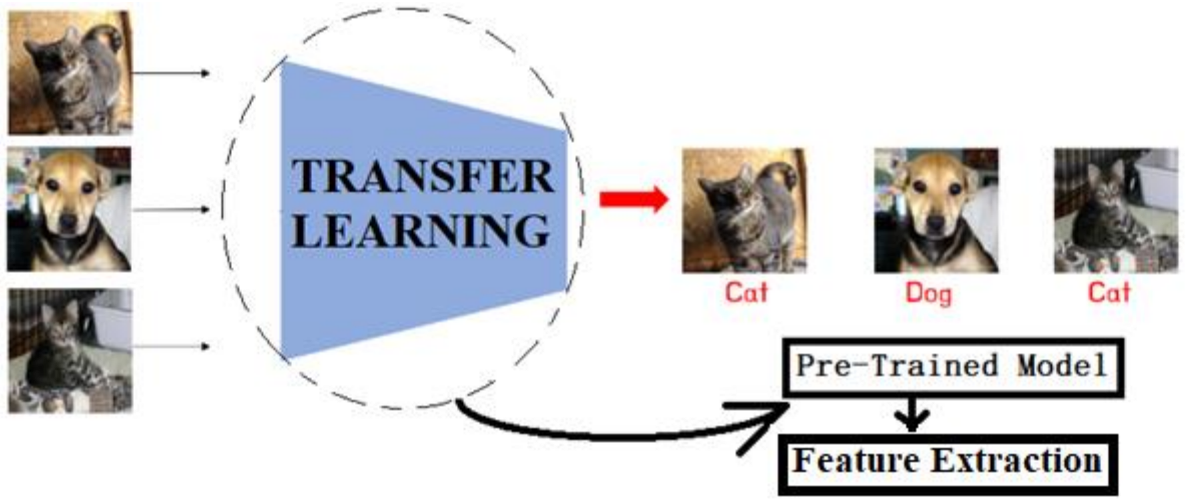
2.2 Transfer Learning

2.2.1 How it Works?

Transfer learning is sort of learning method that train with huge dataset in advance, then replace the output layer with our purpose. Fine-tuning and Feature Extraction are two different approaches used in transfer learning.

- ✓ In feature extraction, we start with a pretrained model and only update the final layer weights from which we derive predictions.
- ✓ In fine-tuning, we start with a pretrained model and update all of the model's parameters for our new task, in essence retraining the whole model.

So here we will be using part of a pre-trained image classifier models (trained on **ImageNet**) as a **feature extractor**, and train additional new layers to perform the cats and dogs classification task



2.2.2 Pre-Trained Models

A pre-trained model is a saved network that was previously trained on a large dataset, typically on a large-scale image-classification task.

For instance, MobileNet is well-trained with ImageNet dataset, but our goal is to classify just two classes, cats and dogs.

So we tested various models, based on their Memory Size, Accuracy and Inference time sourcing from following

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

Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
Xception	88	79.0%	94.5%	22.9M	81	109.4	8.1
VGG16	528	71.3%	90.1%	138.4M	16	69.5	4.2
VGG19	549	71.3%	90.0%	143.7M	19	84.8	4.4
ResNet50	98	74.9%	92.1%	25.6M	107	58.2	4.6
ResNet101	171	76.4%	92.8%	44.7M	209	89.6	5.2
ResNet152	232	76.6%	93.1%	60.4M	311	127.4	6.5
InceptionV3	92	77.9%	93.7%	23.9M	189	42.2	6.9
InceptionResNetV2	215	80.3%	95.3%	55.9M	449	130.2	10.0
MobileNet	16	70.4%	89.5%	4.3M	55	22.6	3.4
MobileNetV2	14	71.3%	90.1%	3.5M	105	25.9	3.8
DenseNet121	33	75.0%	92.3%	8.1M	242	77.1	5.4
DenseNet169	57	76.2%	93.2%	14.3M	338	96.4	6.3
DenseNet201	80	77.3%	93.6%	20.2M	402	127.2	6.7
NASNetMobile	23	74.4%	91.9%	5.3M	389	27.0	6.7
EfficientNetB0	29	77.1%	93.3%	5.3M	132	46.0	4.9
EfficientNetB1	31	79.1%	94.4%	7.9M	186	60.2	5.6
EfficientNetB2	36	80.1%	94.9%	9.2M	186	80.8	6.5
EfficientNetB3	48	81.6%	95.7%	12.3M	210	140.0	8.8
EfficientNetB4	75	82.9%	96.4%	19.5M	258	308.3	15.1

2.3 Sources Used

2.3.1 Framework

PYTORCH:

The PyTorch is an open source machine learning framework based on the Torch library, used for applications such as computer vision, natural language processing and Image Classification primarily developed by Meta AI. It is free and open-source software released under the Modified BSD license.

ADVANTAGES:

- Rich set of powerful APIs to extend the Pytorch Libraries.
- It has computational graph support at runtime.
- It is flexible, faster, and provides optimizations.
- It has support for GPU and CPU.
- Easy to debug using Pythons IDE and debugging tools.

REASON TO CHOOSE: Done with PyTorch Course (by DeepLizard), so felt more comfortable with it.

2.3.2 Development Environment

GOOGLE COLAB

Colab notebooks allow us to combine **Python Executable Code** and **Rich Text** in a single document, along with images, HTML, LaTeX and more. It provides easy-to-configure interactive environment to run your machine learning code that came with access to GPUs for free.

Main features includes High Computational Speed, easy and quick import of packages and even datasets from external sources such as Kaggle, Drive Mounting and Sharing, Code Snippets, Versioning and much more, making it powerful tool to be used in Machine learning.

Anyone can access Google Colab at https://colab.research.google.com

2.4 Code Link

MAJOR:

➢ main.py : https://colab.research.google.com/drive/1r1wyeKY7syziB30ameLnKhVn4NVnPHDP?usp=sharing
➢ API : https://colab.research.google.com/drive/1AhyE52cDKASxaB4nZV1ZmL34Ax9jLpfm?usp=sharing

ADDITIONAL:

✓ ResNet-18 : https://colab.research.google.com/drive/1Q8xmaIMAnMW1DMzUQx0YV9SH4Bk9PFG4?usp=sharing
✓ ResNet-50 : https://colab.research.google.com/drive/1HJYMstRKOL5z2nPvvSl_gvRNqMqjjcBC?usp=sharing
✓ VGG-16 : https://colab.research.google.com/drive/1cwLprw_0RjEzmDrBWkOeTd9GbMpjaQpj?usp=sharing
✓ EfficientNet B0 : https://colab.research.google.com/drive/1zNzqegcTSfn2ELH9s0A0eo9bzD-1w18k?usp=sharing
✓ Inception v3 : https://colab.research.google.com/drive/1lpbZhdlDQz6HidKGFpU6cRZ6ynonpEUH?usp=sharing
✓ Xception : https://colab.research.google.com/drive/1FuRdEHmSCZoz9urTyA4RXkTh18Mv0B_7?usp=sharing
✓ Quant.ResNet-18: https://colab.research.google.com/drive/12xkWARk20ad8Yyu6oePJsWZzRlRfKuXZ?usp=sharing

UNIT: 03 PROCESSING DATASET

Since PyTorch framework is more comfortable to handle on Image dataset, so converting arrays into PNG (High-quality graphics format) formatted images

Instead of having separate files for labels, labels 'Cat.{i}/Dog.{i}', are kept as images name. (Here 'I' can be any number ranging 0-299)

Make sure to upload given numpy arrays, and make folder 'data' with subfolders 'train', 'valid' and 'test' in directory you want.

```
#Importing Necessary Packages
import cv2
import numpy as np
```

3.1 Training Data

```
#TRAINING DATA
numpy_data = np.load("/content/images_train.npy")
numpy_target = np.load("/content/labels_train.npy")
for i in range(0, len(numpy_data)):
  img = numpy_data[i]
  if numpy_target[i] == 0:
   # Till here numpy array, and after here converts to image
    cv2.imwrite(f'/content/data/train/dog.{i}.png', img)
  else:
    cv2.imwrite(f'/content/data/train/cat.{i}.png', img)
```

3.2 Validation Data

```
#VALIDATION DATA
numpy_data = np.load("/content/images_valid.npy")
numpy_target = np.load("/content/labels_valid.npy")
for i in range(0, len(numpy_data)):
  img = numpy_data[i]
  if numpy_target[i] == 0:
   # Till here numpy array, and after here converts to image
    cv2.imwrite(f'/content/data/valid/dog.{i}.png', img)
  else:
    cv2.imwrite(f'/content/data/valid/cat.{i}.png', img)
```

3.3 Testing Data

Here, in Testing Data, no need to mention 'Cat/Dog'; just name with number ranging (0 – 299).

```
#TESTING DATA (labels in images name woouldn't be marked)
numpy_data = np.load("/content/images_test.npy")
numpy_target = np.load("/content/labels_test.npy")
for i in range(0, len(numpy_data)):
  img = numpy_data[i]
  cv2.imwrite(f'/content/data/test1/{i}.png', img)
```

UNIT: 04 MODEL BUILDING

4.1 Import Necessary Libraries:

At first, we need to import some packages for implementation.

```
#Importing Libraries
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os, glob, time, copy, random, zipfile
import matplotlib.pyplot as plt
from PIL import Image
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score, recall_score,precision_score
from tqdm import tqdm_notebook as tqdm
%matplotlib inline
```

```
#Importing PYTORCH Framework Libraries
import torch
import torch.nn as nn
import torch.optim as optim
import torch.utils.data as data
from torchvision.utils import make_grid
import torch.nn.functional as F
import torchvision
from torchvision import models, transforms
```

In order to check the PyTorch's current version use the following command.

```
torch.__version__
```

```
>>>>>>> 1.12.1+cu113
```

4.2 Prepare Dataset

Proper dataset preparation and loading complete data in required format is the most attention-demanding process in Machine Learning following the correct initialized variables, if any.

4.2.1 Importing the Preprocessed Datasets

Make sure to mount drive, if you saved your preprocessed dataset on your drive before.

```
#Directing directives of stored preprocessed data
train_path = '/content/drive/MyDrive/data/train'
valid_path = '/content/drive/MyDrive/data/valid'
test_path = '/content/drive/MyDrive/data/test'
```

4.2.2 Format Verification and Listing

As discussed before, dataset has been preprocessed into ‘.png’ formatted images. Do insure to check and load for only ‘.png’ images in order to bypass Google Colab generated **checkpoints**.

Initialize variables containing list of images for better processing and computational speed

```
#Checking for only 'png' formatted
train_list = glob.glob(os.path.join(train_path, '*.png'))
valid_list = glob.glob(os.path.join(valid_path, '*.png'))
test_list = glob.glob(os.path.join(test_path, '*.png'))
```

4.2.3 Datasets Visualization

Let us visualize some random images from all datasets; namely ‘Train’, ‘Valid’ and ‘Test, in order to insure preprocessed image dataset and correct loading and listing.

Here loop is used for the purpose of displayed different images of cats and dogs, each time you run the cell.

```
#Train Dataset
_, _, files = next(os.walk(train_path))
file_count = print('TOTAL TRAIN IMAGES ARE : {}'.format(len(files)))
r = list(range(len(files)//2))
random.shuffle(r)
j=0
fig = plt.figure(figsize=(30,10))
for i in r:
    img_path = os.path.join(train_path, 'cat.{}.png'.format(i))
    if (os.path.exists(img_path )) == True:
       img = Image.open(img_path)
    else:
       img_path = os.path.join(train_path, 'dog.{}.png'.format(i))
       img = Image.open(img_path)
    j+=1
    plt.subplot(3,8, j)
    plt.imshow(img)
    if j>=24: break
plt.show()
```

```
#Validation Dataset
_, _, files = next(os.walk(valid_path))
file_count = print('TOTAL Validation IMAGES ARE : {}'.format(len(files)))
r = list(range(len(files)//2))
random.shuffle(r)
j=0
fig = plt.figure(figsize=(30,10))
for i in r:
    img_path = os.path.join(valid_path, 'cat.{}.png'.format(i))
    if (os.path.exists(img_path )) == True:
       img = Image.open(img_path)
    else:
       img_path = os.path.join(valid_path, 'dog.{}.png'.format(i))
       img = Image.open(img_path)
    j+=1
    plt.subplot(3,8, j)
    plt.imshow(img)
    if j>=24: break
plt.show()
```

```
#Test Dataset
_, _, files = next(os.walk(test_path))
file_count = print('TOTAL TEST IMAGES ARE : {}'.format(len(files)))
r = list(range(len(files)))
random.shuffle(r)
j=0
fig = plt.figure(figsize=(30,10))
for i in r:
    img_path = os.path.join(test_path, '{}.png'.format(i))
    img = Image.open(img_path)
    j+=1
    plt.subplot(3,8, j)
    plt.imshow(img)
    if j>=24: break
plt.show()
```

4.2.4 Transformation and Augmentation

Mentioned earlier, dataset is released in Kaggle. Original dataset has 12500 images of dogs and 12500 images of cats, in 25000 images in total. That's a huge amount to train the model. But in our case, we just only use 600 images for training, 300 images for validation, and 300 images for test.

Actually, 300 images are not enough datasets for training. But we can size-up the dataset with transformation. Yes, it is **data augmentation**. There are several techniques to transform the image. In this case, we will use following transformations:

- Resize: Of course, model input must be the same size .So we need to resize our transformed image to fixed size. This fixed size gets declared by variable, defined later, based upon model selected.
- Random Affine: We can use linear mapping method by preserving points, straight lines, and planes; comes from TorchVision.
- Random Horizontal Flips: We can horizontally flip a targeted image with a given probability.

One more API we need to implement is normalize. Normalization is one of method for rescaling. There are several techniques for normalization. But in this API, our normalize function will be based upon mean standard deviation values.

NOTE:

- ✓ If we use .Normalize((0, 0, 0), (1, 1,1)) instead of .Normalize (mean, std), we found to loss accuracy, both train and test, by 2 – 4%
- ✓ Same transformation will be applied on Testing Data as now declared for Validation Data.

```
#Defining Two Transformation (Testing will also use that of Validation)
class CustomTransform():

    def __init__(self, resize, mean, std):
        self.data_transform = {
            'train': transforms.Compose([
        transforms.Resize(size),
        transforms.RandomAffine(0, shear=10, scale=(0.8,1.2)),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize(mean, std)
    ]),
            'valid':   transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(size),
        transforms.ToTensor(),
        transforms.Normalize(mean, std)
    ]),     }
    def __call__(self, img, phase):
        return self.data_transform[phase](img)
```

4.2.5 Retrieval of Images and Labels

Usually dataset pipeline is built for training and test dataset. Actually, it is very efficiency for memory usage, because its type is python generator. We are done with it in Preprocessed step as following,

- ❖ DATA
    - Train
    - Valid
    - Test

Now we need to extract parent name as a target, to be later called by dataloader.

```
class DogCatDataset(data.Dataset):

    def __init__(self, file_list, transform=None, phase='train'):
        self.file_list = file_list
        self.transform = transform
        self.phase = phase

    def __len__(self):
        return len(self.file_list)

    def __getitem__(self, i
        image_loc = self.file_list[idx]
        image = Image.open(image_loc)

        img_transformed = self.transform(image, self.phase)

        # Get Target
        target = image_loc.split('/')[-1].split('.')[0]
        if      target == 'dog':       target = 0
        elif    target == 'cat':       target = 1
        else:   print('Wrong DATASET')

        return img_transformed, target
```

4.2.6 Hyper Parameters

Tuned via performing various trainings and tests, here all models input (224 x 224) pixels image.

```
#Defining Variables
mean = (0.485, 0.456, 0.406)
std = (0.229, 0.224, 0.225)
size = 224
batch = 32   #Gives best result due to same Tensor .dtype
```

4.2.7 GPU Availability

Let’s check for availability of GPU.GPU helps to perform the training on large and complex datasets with ease as compared to CPU. So availability of GPU gives advantage on faster training of the model.

Since Google Colab provides a free **12GB NVIDIA Tesla K80** GPU to run up to 12 hours continuously, so using following command makes the code much flexible with respect to compute resources.

But here all the testing is performed on free version of Google Colab, so in order to prevent disturbance in runtime, hardware accelerator is set to ‘default’, setting to GPU or TPU have significant impact on processing time

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

4.2.8 Data Loader

Load the Training and Validation Dataset by calling previous made classes and print some check-marks for verification.

```
#Making separate Datasets by call Custom Dataset Class
train_dataset = DogCatDataset(train_list, transform=CustomTransform(size,
mean, std), phase='train')
valid_dataset = DogCatDataset(valid_list, transform=CustomTransform(size,
mean, std), phase='valid')

# Verifivation
print('Verification')
index = 0
print(train_dataset.__getitem__(index)[0].size())
print(train_dataset.__getitem__(index)[1])
```

```
#DataLoading (For Pytorch FrameWork)
train_dataloader = data.DataLoader(train_dataset, batch_size=batch, shuffl
e=True)
valid_dataloader = data.DataLoader(valid_dataset, batch_size=batch, shuffl
e=False)

#Dictionary for Simplifying and Compacting Code
loaderDict = {'train': train_dataloader, 'valid': valid_dataloader}

# Verification
print('Verification')
batch_iterator = iter(train_dataloader)
inputs, target = next(batch_iterator)
print(inputs.size())
print(target)
```

4.2.9 Check Functionality of Data Transformation:

We have successfully applied data transformation, made the batches and loaded it to the data loader. Now, let's make a check point here, and visualize some random loaded train and validation data, in form of batches, for idea

```
for images, labels in train_dataloader:

    fig, ax = plt.subplots(figsize = (40, 10))
    ax.set_xticks([])
    ax.set_yticks([])
    ax.imshow(make_grid(images, 8).permute(1,2,0))
    break
```

```
for images, labels in valid_dataloader:

    fig, ax = plt.subplots(figsize = (40, 10))
    ax.set_xticks([])
    ax.set_yticks([])
    ax.imshow(make_grid(images, 4).permute(1,2,0))
    break
```

4.3 Training

Let’s now move towards most crucial part of Machine Learning, of course training and validation part.

4.3.1 Definition:

A function has been defined for model training, taking four arguments; pre-trained model to be used, criterion and optimizer mainly depending upon model selected and number of epochs specified by user.

```
def train_model(model, loaderDict, criterion, optimizer, num_epoch):

    since = time.time()
    best_model_wts = copy.deepcopy(net.state_dict())
    accuracy  = 0.0
    model = model.to(device
    for epoch in range(num_epoch):
        print('Epoch {}/{}'.format(epoch + 1, num_epoch))
        print('-'*20)
        for phase in ['train', 'valid']:
            if phase == 'train':
                model.train()
            else:
                model.eval()

            epoch_loss = 0.0
            epoch_corrects = 0
            for inputs, target in tqdm(loaderDict[phase]):
                inputs = inputs.to(device)
                target = target.to(device)
                optimizer.zero_grad()
                with torch.set_grad_enabled(phase == 'train'):
                    outputs = net(inputs)
                    _, preds = torch.max(outputs, 1)
                    loss = criterion(outputs, target)
                    if phase == 'train':
                        loss.backward()
                        optimizer.step()
                    epoch_loss += loss.item() * inputs.size(0)
                    epoch_corrects += torch.sum(preds == target.data)

            epoch_loss = epoch_loss / len(loaderDict[phase].dataset)
            epoch_acc = epoch_corrects.double() / len(loaderDict[phase].d
ataset)

            print('{} Loss: {:.4f} Acc: {:.4f}'.format(phase, epoch_loss,
 epoch_acc))
            if phase == 'valid' and epoch_acc > accuracy :
                accuracy  = epoch_acc
                best_model_wts = copy.deepcopy(model.state_dict())
    time_elapsed = time.time() - since
    print('Training complete in {:.0f}m {:.0f}s'.format(
        time_elapsed // 60, time_elapsed % 60))
    print('High Achieved validation Accuracy: {:4f}'.format(accuracy ))

    np.save('trainloss.npy',train_losses)
    np.save('_validloss.npy',valid_losses)

    # load best model weights
    model.load_state_dict(best_model_wts)
    return model
```

4.3.2 Test Some Pre-Trained Models

We don't want to retrain whole specified Conv layers, since they are already trained with ImageNet dataset. But we need to **update** the Dense layer **weights** for classifying cats and dogs. This kind of process is called **Feature Extraction**.

- ✓ For Fine-Tuning, put `param.requires_grad = True.`
- ✓ For Feature Extraction, put `param.requires_grad = False.`

In short, for feature extraction we need to modify dense layer according to own requirements.

CAUTION: Do run and test one pre-trained model at a time.

4.3.2.1 MobileNet_V2

MobileNet-v2 is a convolutional neural network that is **53 layers deep**, used for multiple use cases. Depending on the use case, it can use different input layer size and different width factors. This allows different width models to reduce the number of multiply-adds and thereby reduce inference cost, mainly designed for mobile devices.

```
#Defining PreTrained MobileNet V2 model
net = models.mobilenet_v2(pretrained=True).to(device)

for param in net.parameters():
    param.requires_grad = False

net.classifier = nn.Sequential(
    nn.Linear(1280, 128, bias=True),
    nn.ReLU(inplace=True),
    nn.Linear(128, 2)).to(device)

#Defining Loss Function and Optimizer
optimizer = optim.Adam(net.parameters())
criterion = nn.CrossEntropyLoss()
num_epoch = 15
net_trained = train_model(net, loaderDict, criterion, optimizer, num_epoc
h)
torch.save(net_trained, '/content/drive/MyDrive/data/mobilenet.pt')
model_saved = torch.load('/content/drive/MyDrive/data/mobilenet.pt')
```

4.3.2.2 ResNet-18

ResNet-18 is a convolutional neural network, a **72-layer architecture with 18 layers deep**, used classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals.

```
 #Defining PreTrained Resnet18 model
net = models.resnet18(pretrained=True).to(device)

for param in net.parameters():
    param.requires_grad = False

net.fc = nn.Sequential(
    nn.Linear(in_features=512, out_features=1000, bias=True),
    nn.ReLU(inplace=True),
    nn.Linear(1000, 2)).to(device)
#Defining Loss Function and Optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters())

num_epoch = 15

net_trained = train_model(net, loaderDict, criterion, optimizer, num_epoc
h)
torch.save(net_trained, '/content/drive/MyDrive/data/resnet18.pt')
model_saved = torch.load('/content/drive/MyDrive/data/resnet18.pt')
```

4.3.2.3 ResNet-50

ResNet-50 is a convolutional neural network that is **50 layers deep**. ResNet, short for Residual Networks is a classic neural network used as a backbone for many computer vision tasks.

```
 #Defining PreTrained Resnet50 model
net = models.resnet50(pretrained=True).to(device)

for param in net.parameters():
    param.requires_grad = False

net.fc = nn.Sequential(
               nn.Linear(2048, 128),
               nn.ReLU(inplace=True),
               nn.Linear(128, 2)).to(device)
#Defining Loss Function and Optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters())

num_epoch = 15

net_trained = train_model(net, loaderDict, criterion, optimizer, num_epoc
h)
torch.save(net_trained, '/content/drive/MyDrive/data/resnet50.pt')
model_saved = torch.load('/content/drive/MyDrive/data/resnet50.pt')
```

4.3.2.4 VGG-16

VGG-16 is a convolutional neural network that is 16 layers deep, a pretty extensive network and has a total of around 138 million parameters. It was used to win ILSVR (Imagenet) competition in 2014. It is considered to be one of the excellent vision model architecture.

```
 #Defining PreTrained VGG16 model
net = models.vgg16(pretrained=True).to(device)

for param in net.parameters():
    param.requires_grad = False

net.classifier = nn.Sequential(
    nn.Linear(in_features=25088, out_features=4096, bias=True),
    nn.ReLU(inplace=True),
    nn.Dropout(p=0.5, inplace=False),
    nn.Linear(in_features=4096, out_features=4096, bias=True),
    nn. ReLU(inplace=True),
    nn.Dropout(p=0.5, inplace=False),
    nn.Linear(4096, 128, bias=True),
    nn.ReLU(inplace=True),
    nn.Linear(128, 2)).to(device)


criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters())
net_trained = train_model(net, loaderDict, criterion, optimizer, num_epoc
h)

torch.save(net_trained, '/content/drive/MyDrive/data/VGG.pt')
model_saved = torch.load('/content/drive/MyDrive/data/VGG.pt')
```

4.3.2.5 EfficientNetB0

EfficientNet-B0 is a convolutional neural network, with total number of layers **237**, used to classify more than million images into 1000 object categories.

EfficientNet-B0 is not pure PyTorch's built-in model, so can only be called as given below.

Use command, `!pip install efficientnet_pytorch` ,everytime you are going to use any EfficientNet.

```
def create_params_to_update(net):

    params_to_update_1 = []
    update_params_name_1 = ['_fc.weight', '_fc.bias']

    for name, param in net.named_parameters():
        if name in update_params_name_1:
            param.requires_grad = True
            params_to_update_1.append(param)
            #print("{} 1".format(name))
        else:
            param.requires_grad = False
            #print(name)

    params_to_update = [
        {'params': params_to_update_1}
    ]

    return params_to_update
```

```
 #Defining PreTrained EfficientNet B0 model
!pip install efficientnet_pytorch
from efficientnet_pytorch import EfficientNet
net = EfficientNet.from_pretrained('efficientnet-b0')

net._fc = nn.Linear(in_features=1280, out_features=2)
params_to_update = create_params_to_update(net)
optimizer = optim.Adam(params=params_to_update, weight_decay=1e-4)

criterion = nn.CrossEntropyLoss()
net_trained = train_model(net, loaderDict, criterion, optimizer, num_epoc
h)

torch.save(net_trained, '/content/drive/MyDrive/data/efficientnetb0

.pt')
model_saved = torch.load('/content/drive/MyDrive/data/efficientnetb0

.pt')
```

4.3.2.6 Inception v3

The Inception v3 model was released in the year 2015, it has a total of 42 layers and a lower error rate than its predecessors. The second output is known as an auxiliary output and is contained in the **AuxLogits** part of the network. The primary output is a linear layer at the end of the network.

Inception v3 strictly inputs 299 x 299 images, so modify **size = 229** in Hyperparameters.

```
 #Defining PreTrained Inceptionv3 model
net  = models.inception_v3(pretrained=True).to(device)
params_to_update = []
for name,param in net.named_parameters():
    if param.requires_grad == True:
        params_to_update.append(param)

net.aux_logits=False

net.fc = nn.Sequential(
               nn.Linear(2048, 128),
               nn.ReLU(inplace=True),
               nn.Linear(128, 2)).to(device)
#Defining Loss Function and Optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(params_to_update)
net_trained = train_model(net, loaderDict, criterion, optimizer, num_epoch)

torch.save(net_trained, '/content/drive/MyDrive/data/inceptionv3.pt')
model_saved = torch.load('/content/drive/MyDrive/data/ inceptionv3.pt')
```

4.3.2.7 Xception

Xception is a convolutional neural network that is 71 layers deep, used to classify images into 1000 categories.

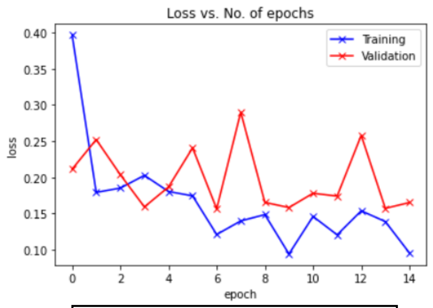
Use command, **!pip install timm ,** everytime you use Xception model

```
 #Defining PreTrained XCEPTION model
import timm

net = timm.create_model('xception', pretrained=True)
for param in net.parameters():
    param.requires_grad = False

net.fc = nn.Sequential(
               nn.Linear(2048, 128),
               nn.ReLU(inplace=True),
               nn.Linear(128, 2)).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters())

net_trained = train_model(net, loaderDict, criterion, optimizer, num_epoch)

torch.save(net_trained, '/content/drive/MyDrive/data/xception.pt')
model saved = torch.load('/content/drive/MyDrive/data/xception.pt')
```

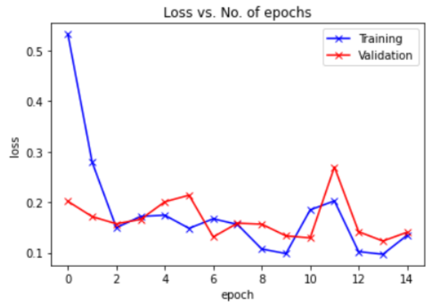
4.3.3 Comparison

For sake of comparison between losses of different models, arrays containing **Training and Validation Losses** have been stored to working directory and can be displayed as running following cell.

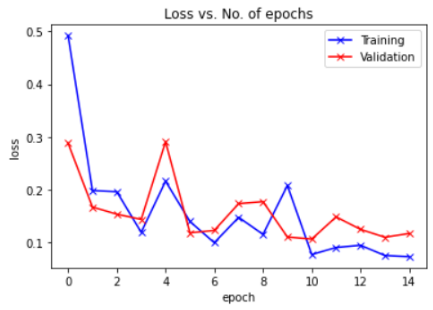
```
train_loss = np.load('trainloss.npy')
valid_loss= np.load('validloss.npy')
plt.plot(train_loss, '-bx')
plt.plot(valid_loss, '-rx')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend(['Training', 'Validation'])
plt.title('Loss vs. No. of epochs');
```



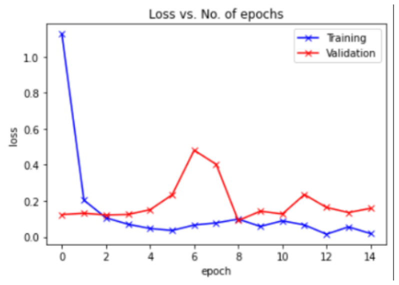
MobileNet v2



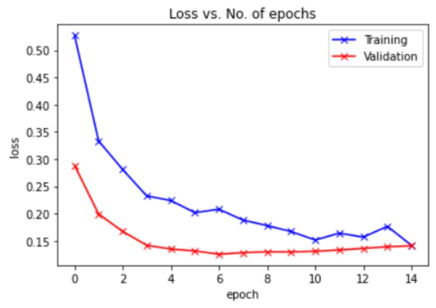
ResNet 18



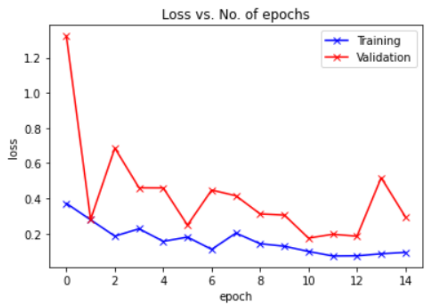
ResNet 50



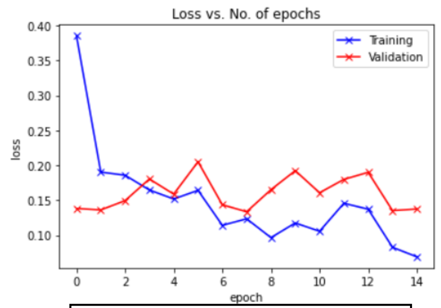
VGG-16



EfficientNetB0



Inception v3



Xception

The Addition of more Deep Layers to any of Testing Network results in a Degradation of the Output.

4.4 Testing

4.4.1 Prediction Call:

Now, test the built model on Test Data, store the ID and predicted target as **Pandas Dataframe,** and save the final report as ‘.csv’ format

```
# Prediction
id_list = []
propabability_list = []

with torch.no_grad():
    for test_loc in tqdm(test_list):
        img = Image.open(test_loc)
        _id = int(test_loc.split('/')[-1].split('.')[0])

        transform = CustomTransform(size, mean, std)
        img = transform(img, phase='valid')
        img = img.unsqueeze(0)
        img = img.to(device)

        model_saved.eval()

        outputs = model_saved(img)
        propabability = F.softmax(outputs, dim=1)[:, 1].tolist()

        id_list.append(_id)
        propabability_list.append(propabability[0])


dataFrame = pd.DataFrame({
    'id': id_list,
    'probability': propabability_list
})

dataFrame.sort_values(by='id', inplace=True)
dataFrame.reset_index(drop=True, inplace=True)

dataFrame.to_csv('evaluation_report.csv', index=False)
```

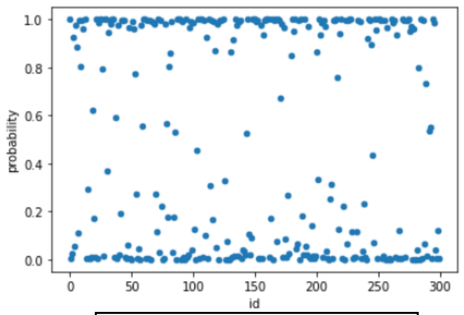
4.4.2 Predicted DataFrame Visualization:

Predicted dataframe has been previously saved on working directory as **evaluation_report.csv,** now in order to access and read dataframe, use following commands:

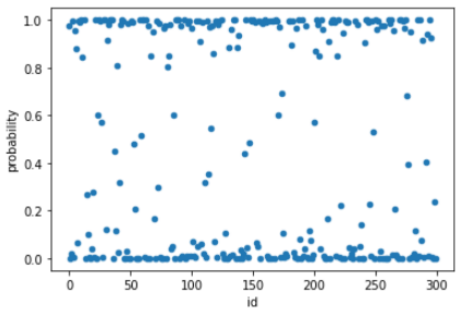
```
display(dataFrame)
```

For visualizing dataframe in form of Scatter plot, use

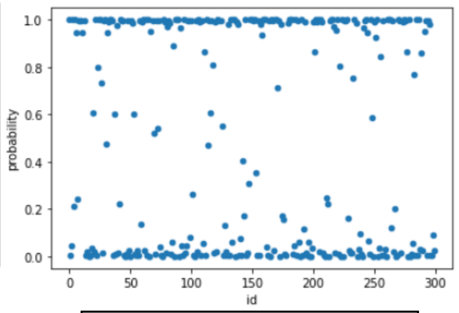
```
dataFrame.plot(
    x = 'id',
    y='probability',
    kind='scatter'     )
plt.show()
```



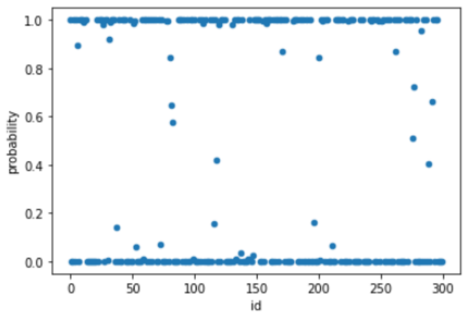
MobileNet_V2



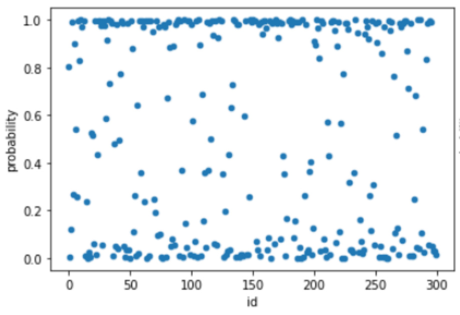
Resnet18



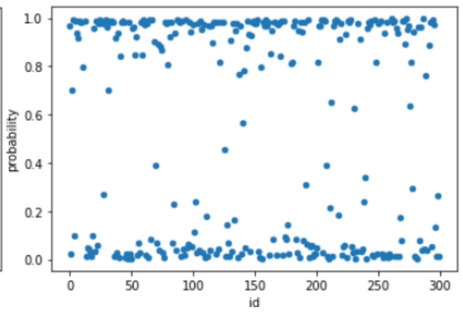
Resnet50



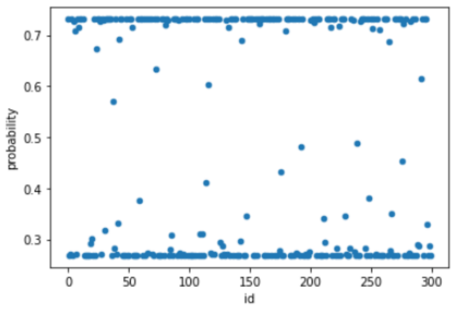
VGG 16



EfficientNetB0



Inception_V3



Xception

The Cleanest Scattar Plot has been given by VGG-16.

4.4.3 Binary Classification:

Set the threshold, and make array of predicted target either as 0 or 1.

```
temp = (dataFrame["probability"])
y_pred = np.zeros(300, dtype=int)
for i in range(len(temp)):
  if temp[i] > 0.56:
   y_pred[i] = 1
#Verification
print(y_pred)
```

4.4.4 Comparison with Given Test Targets:

Upload the given Test Labels array and **assert** if length of actual labels array is not equal to predicted labels array.

```
y_actual = np.load('/content/labels_test.npy')
print(y_actual)
assert ((len(y_actual)) != (len(y_pred)), f"MISMATCHED LENGTHS" )
```

It's time to visual the wrong predicted images by model by comparison between arrays.

```
mismatch = 0
id_list = []
fig = plt.figure(figsize=(12,12))
j=1
class_ = {1: 'CAT ', 0: 'DOG '}
for i in range(len(y_actual)):
  if y_actual[i] != y_pred[i]:
    img_path = os.path.join(test_path, '{}.png'.format(i))
    img = Image.open(img_path)
   # print(y_pred[i])
   # print(temp[i])
    plt.subplot(3, 3, j)
    plt.title(str(class_[y_pred[i]]) + "  Probability :{:4f}".format(temp
[i]))
    plt.imshow(img)
    mismatch+=1
    j+=1
plt.show()
```

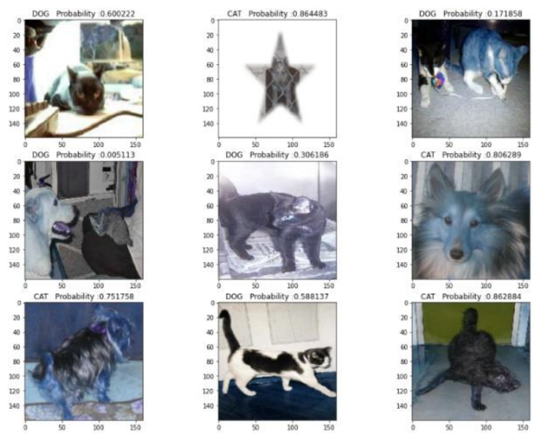
WRONG PREDICTED IMAGES

CAT Probability :0.855118
DOG Probability :0.519196
DOG Probability :0.122274
DOG Probability :0.085575
DOG Probability :0.238861
CAT Probability :0.917304
DOG Probability :0.448388
DOG Probability :0.457584
DOG Probability :0.448971

MobileNet v2

CAT Probability :0.451675
CAT Probability :0.517389
CAT Probability :0.544132
CAT Probability :0.908788
DOG Probability :0.118162
DOG Probability :0.405597
DOG Probability :0.045244

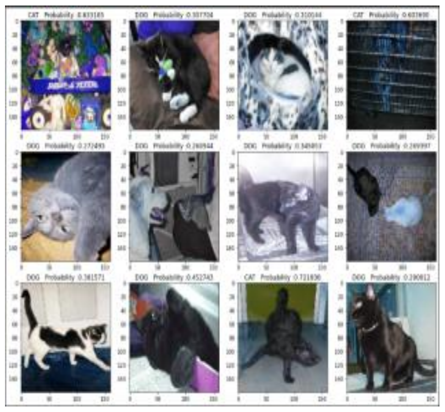
ResNet 18


DOG Probability :0.600222
CAT Probability :0.864483
DOG Probability :0.171858
DOG Probability :0.005113
DOG Probability :0.306186
CAT Probability :0.806289
CAT Probability :0.751758
DOG Probability :0.588137
CAT Probability :0.862884

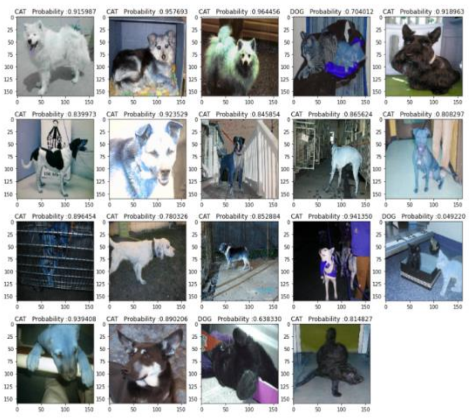
ResNet 50

DOG Probability :0.540446
DOG Probability :0.437161
DOG Probability :0.113777
CAT Probability :0.641021
DOG Probability :0.434862
CAT Probability :0.727650
DOG Probability :0.014145
DOG Probability :0.259518
DOG Probability :0.306889
CAT Probability :0.714963
DOG Probability :0.541875

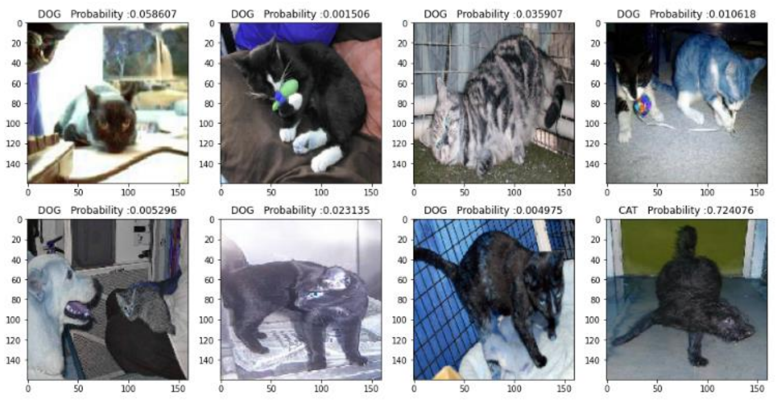
EfficientNet B0



Xception



Inception V3


DOG Probability :0.058607
DOG Probability :0.001506
DOG Probability :0.035907
DOG Probability :0.010618
DOG Probability :0.005296
DOG Probability :0.023135
DOG Probability :0.004975
CAT Probability :0.724076

VGG-16

4.4.5 Check F1_Score, Precision Score, Recall Score

```
print('Total Mismatched : {}'.format(mismatch))
print('F1_SCORE : {:4f}'.format(f1_score(y_actual, y_pred, average="binar
y")))
print('Precision_Score : {:4f}'.format(precision_score(y_actual, y_pred,
average="binary")))
print('Recall_Score : {:4f}'.format(recall_score(y_actual, y_pred, averag
e="binary")))
```

PRE-TRAINED MODELS	Total Mismatched	F1 Score	Precision Score	Recall Score	Inference Time
1. MOBILENET_V2	8	0.9735	0.9671	0.9800	0.053
2. RESNET18	7	0.9767	0.9735	0.9800	0.093
3. RESNET50	9	0.9699	0.9731	0.9666	0.226
4. VGG16	8	0.9727	0.9930	0.9533	0.609
5. EFFICIENTNETB0	11	0.9627	0.9793	0.9466	0.098
6. INCEPTIONV3	19	0.9392	0.9018	0.9800	0.219
7. XCEPTION	12	0.9591	0.9791	0.9400	0.2913

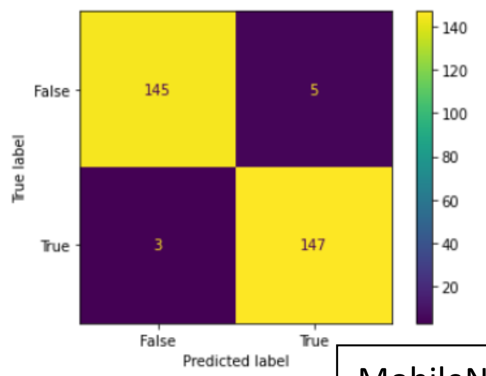
*Inference time values have been taken from **Section 4.4.7**

4.4.6 Confusion Matrix

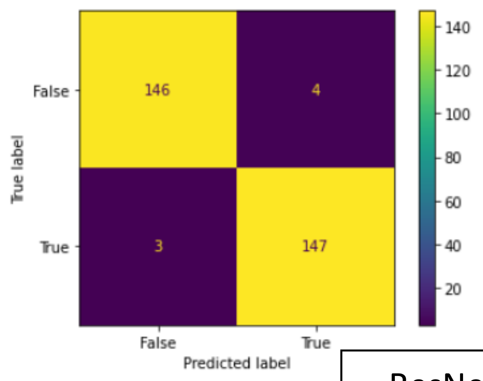
Confusion Matrix is a useful machine learning method which allows you to measure **Recall, Precision, and Accuracy**. Below given can be used to know the terms: True Positive, True Negative, False Negative, and True Negative. True Positive.

```
from sklearn import metrics

confusion_matrix = metrics.confusion_matrix(y_actual, y_pred)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_m
atrix, display_labels = [False, True])
cm_display.plot()
plt.show()
```

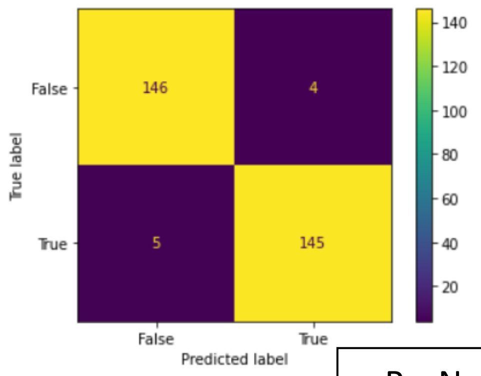
CONFUSION MATRICES OBTAINED



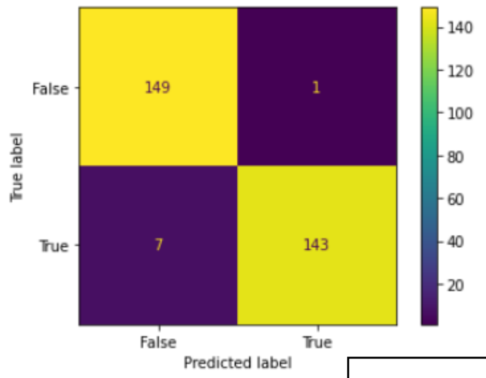
MobileNet v2



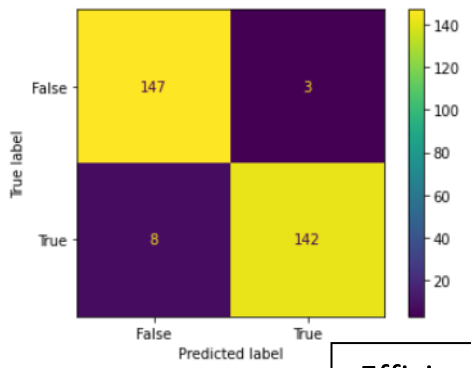
ResNet 18



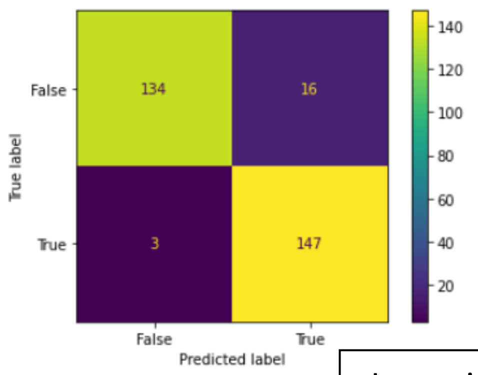
ResNet 50



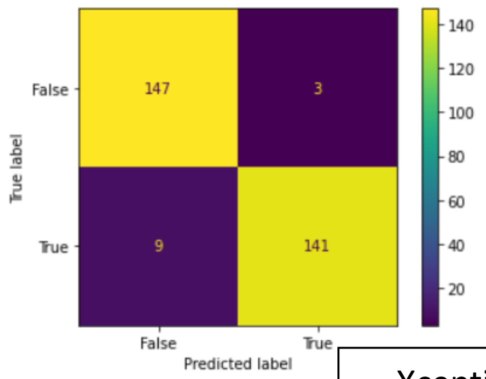
VGG-16



EfficientNet b0



Inception v3



Xception

4.4.7 Inference Time

In deep learning, inference time is the amount of time it takes for a machine learning model to process new data and make a prediction.

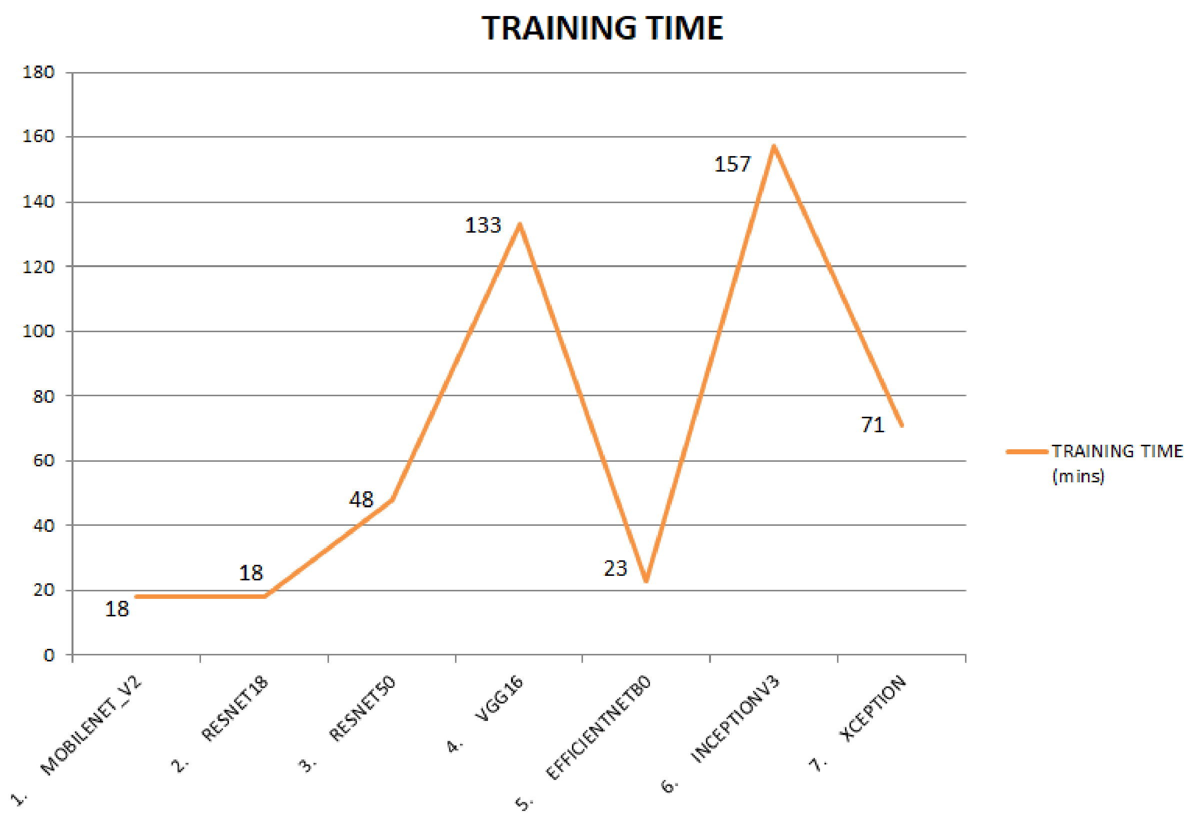
To calculate estimated Inference Time, random 6 images have been taken from **Google Search,** uploaded on Drive and following code is run.

```
#Estimating Inference Time

import torch
import time
from torchvision import transforms
import torch.nn.functional as F
import glob
from PIL import Image

model = torch.load("/content/drive/MyDrive/data/mobilenet.pt")
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
total = 0.0
mean = (0.485, 0.456, 0.406)
std = (0.229, 0.224, 0.225)
transf= transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(mean, std)   ])

for x in glob.glob('/content/drive/MyDrive/Images/*.jpg'):
  image = Image.open(x)
  since = time.time()
  img = transf(image)
  img = img.unsqueeze(0)
  img = img.to(device)
  pred = net_trained(img)
  pred = F.softmax(pred, dim=1)[:, 1]
  temp = pred[0] > 0.5
  if temp==1:
      print("PREDICTION = CAT ")
      print(' Probability : {:4f}'.format(pred.item()))
  else:
      print("PREDICTION = DOG ")
      print(' Probability : {:4f}'.format(1 - pred.item()))
  time_elapsed = time.time() - since
  total = total + time_elapsed

print('\n CALCULATED INFERENCE TIME : {}'.format(total/(float(6))) )
```

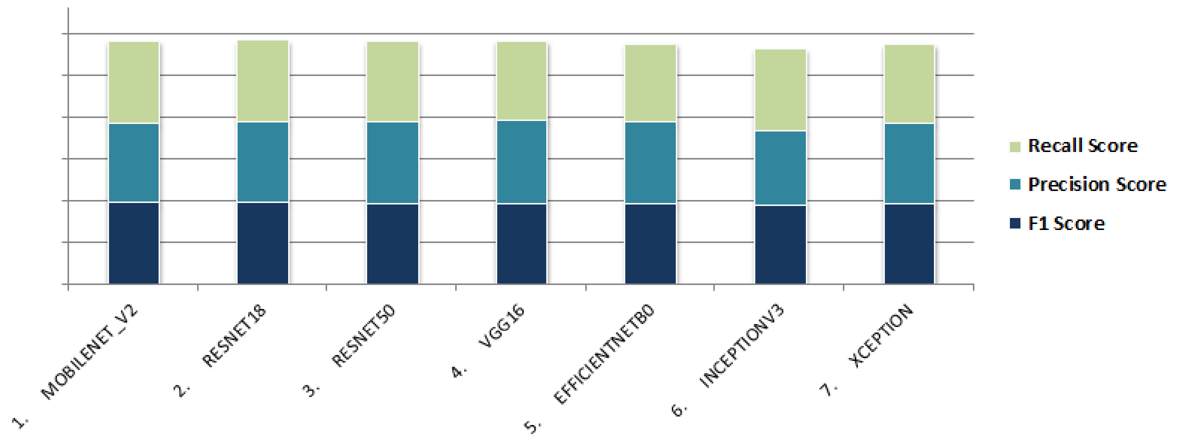
4.5 Conclusion

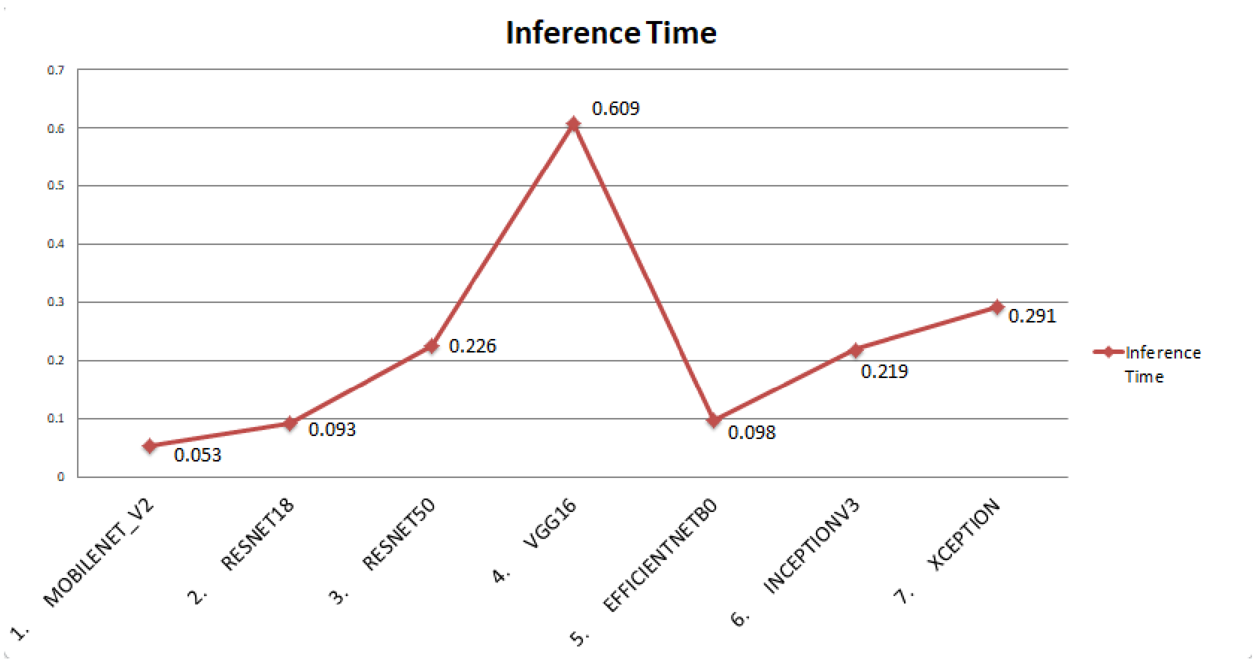
7 Models have been tested, now let’s move to select best model out of all.

4.5.1 Graphical Evidences:



Least Training Time requiring models are MobileNet v2 and ResNet-18, while Inception v3 and VGG-16 taking the greatest time.





MobileNet v2 is taking least inference time, while VGG-16 is taking the greatest.

NOTE: The accuracy of test dataset is generally lower than training/validation accuracy. Because pre-trained models are not always perfect models for classification, and test dataset may be the unknown data for trained model. Maybe the class distribution is different from training set and test set.

4.5.2 Selected Model:

- ✓ MobileNet v2 has been selected as best performing model w.r.t both **Accuracy and Inference Time.**
- ✓ Also we will consider Resnet-18, due to comparable training, inference time and least miscounts and VGG-16, for giving cleanest Scatter Plot.

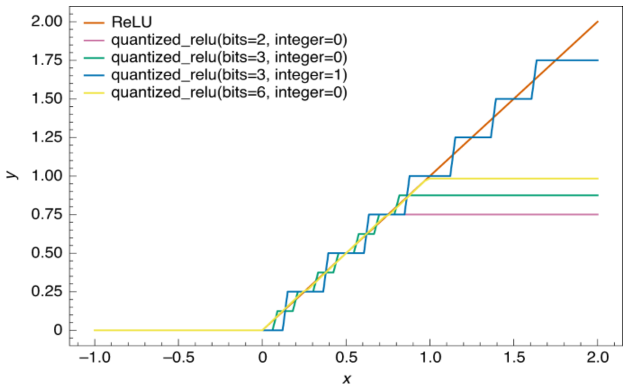
NOTE:

2-3 testings have been performed on individuals and best results taken from that.

UNIT: 05 ADDITIONAL MODEL BUILDING

5.1 What is Quantization?

Quantization in Machine Learning (ML) is the process of converting data in FP32 (floating point 32 bits) to a smaller precision like INT8 (Integer 8 bit) and perform all critical operations like Convolution in INT8 and at the end, convert the lower precision output to higher precision in FP32.



This sounds simple but as you can imagine that converting a floating point to integer results in error that can grow up through calculations, maintaining the accuracy is critical. In Quantization, accuracy is within 1% of the original accuracy.

In terms of performance, as we are dealing with 8 bits instead of 32 bits, we should be theoretically 4 times faster. In real life applications, speedup of at least 2X is observed.

OBJECTIVES:

- Increases Power Efficiency for two reasons: reduced memory access costs and increased compute efficiency.
- Using the lower-bit quantized data requires less data movement, both on-chip and off-chip, which reduces memory bandwidth and saves significant energy.
- Easy to implement on hardware devices, for example **FPGAs** giving least Inference Time.

5.2 Code

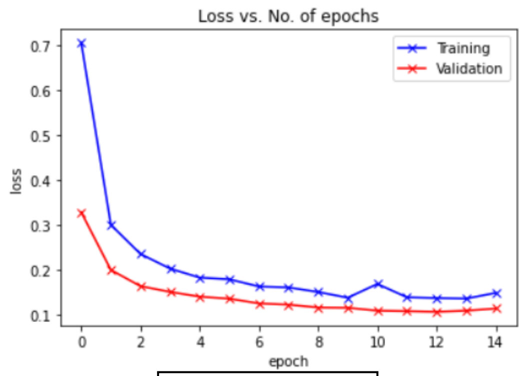
We are going to use PyTorch's built-in Quantized models, but after several testings we found that these built-in models are stable on **Fine Tuning,** but become unstable on **Feature Extraction**. We only found Resnet18 a stable, enough for our comparison with non-Quantized Models

```
model_fe = models.quantization.resnet18(pretrained=True, progress=True, q
uantize=True)
num_ftrs = model_fe.fc.in_features
```

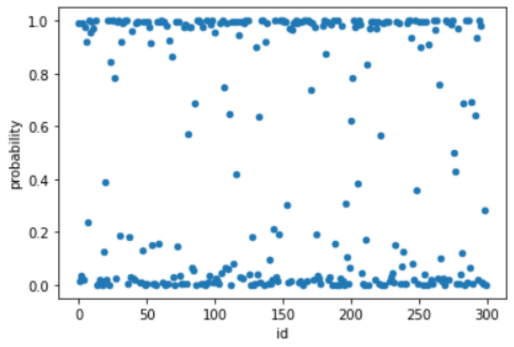
On contradictory to non-Quantized models, where we simply put `param.requires_grad = False` for feature extraction, in Quantized models we manually have to mention which layers we need to freeze and which dense layers for want to add.

```
from torch import nn

def create_combined_model(model_fe):
  # Step 1. Isolate the feature extractor.
  model_fe_features = nn.Sequential(
    model_fe.quant,  # Quantize the input
    model_fe.conv1,
    model_fe.bn1,
    model_fe.relu,
    model_fe.maxpool,
    model_fe.layer1,
    model_fe.layer2,
    model_fe.layer3,
    model_fe.layer4,
    model_fe.avgpool,
    model_fe.dequant,  # Dequantize the output
  )
  # Step 2. Create a new "head"
  new_head = nn.Sequential(
    nn.Dropout(p=0.5),
    nn.Linear(num_ftrs, 2),
  )
  # Step 3. Combine, and don't forget the quant stubs.
  new_model = nn.Sequential(
    model_fe_features,
    nn.Flatten(1),
    new_head,
  )
  return new_model
```

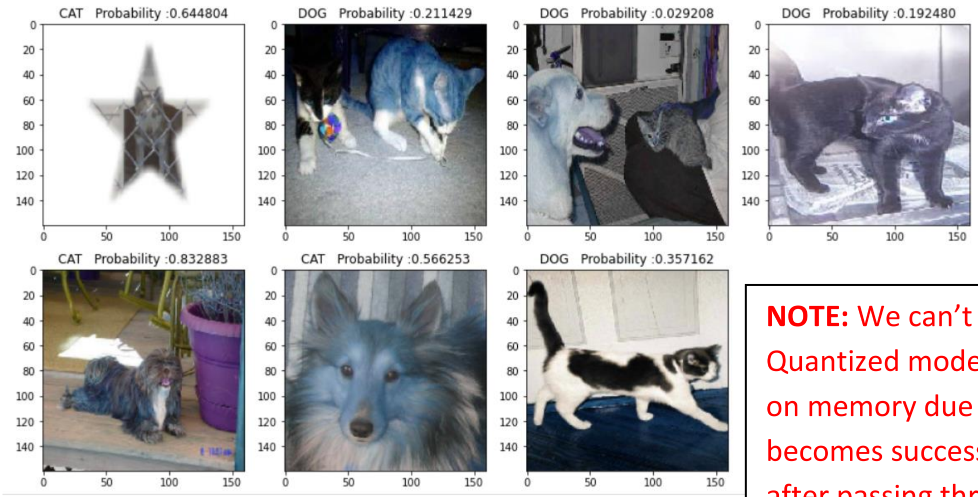
```
net = create_combined_model(model_fe)
net = net.to(device)

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters())

num_epoch = 15
net_trained = train_model(net, loaderDict, criterion, optimizer, num_epoc
h)
```



LOSSES PLOT



SCATTER PLOT


CAT Probability :0.644804
DOG Probability :0.211429
DOG Probability :0.029208
DOG Probability :0.192480
CAT Probability :0.832883
CAT Probability :0.566253
DOG Probability :0.357162

MISMATCHED IMAGES

NOTE: We can't save PyTorch's built-in Quantized model after Feature Extraction on memory due to Stat_Dict () fusion. If becomes successful on saving model after passing through complex process, the model becomes Unstable.

5.3 Result

➢ Training Time :	9.5 minutes
➢ Total Mismatched :	7
➢ F1 Score :	0.9765
➢ Precision and Recall:	0.9798 & 0.9733
➢ Inference Time:	0.055

We found quantized model of even ResNet 18 performed well on gaining higher accuracy and even lesser inference time of MobileNet v2.

If Feature Extraction from **Pre-Trained Model** wasn't the condition, then definitely would go with some Quantized Model.

UNIT: 06 API CONFIGURATION

Using APIs as a clean interface between the analytics and the application that makes use of them allows for faster product development and reusability of developed models in multiple applications.

Streamlit is an awesome new tool that allows engineers to quickly build highly interactive web applications around their data, machine learning models, and pretty much anything, without need of any web development.

6.1 Importing Packages:

```
!pip install -q streamlit
```

```
    |████████████████████████████████| 9.1 MB 8.9 MB/s
    |████████████████████████████████| 235 kB 41.0 MB/s
    |████████████████████████████████| 164 kB 49.1 MB/s
    |████████████████████████████████| 78 kB 6.6 MB/s
    |████████████████████████████████| 4.7 MB 41.3 MB/s
    |████████████████████████████████| 181 kB 44.5 MB/s
    |████████████████████████████████| 63 kB 1.3 MB/s
    |████████████████████████████████| 51 kB 5.0 MB/s

  Building wheel for validators (setup.py) ... done
```

```
!pip install pyngrok
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/

Collecting pyngrok
  Downloading pyngrok-5.1.0.tar.gz (745 kB)
     |████████████████████████████████| 745 kB 6.9 MB/s
Requirement already satisfied: PyYAML in /usr/local/lib/python3.7/dist-packages (from pyngrok) (6.0)
Building wheels for collected packages: pyngrok
  Building wheel for pyngrok (setup.py) ... done
  Created wheel for pyngrok: filename=pyngrok-5.1.0-py3-none-any.whl size=19007 sha256=5edc2be506cfa71f1fac23b66d516508e7a975e4ec81c2a88c79b4329d7a4e3f
  Stored in directory: /root/.cache/pip/wheels/bf/e6/af/ccf6598ecefecd44104069371795cb9b3afbcd16987f6ccfb3
Successfully built pyngrok
Installing collected packages: pyngrok
Successfully installed pyngrok-5.1.0
```

6.2 Setting Front-End:

Make sure to create a file named ‘streamlit_app,py’ in the working directory and then run the following code.

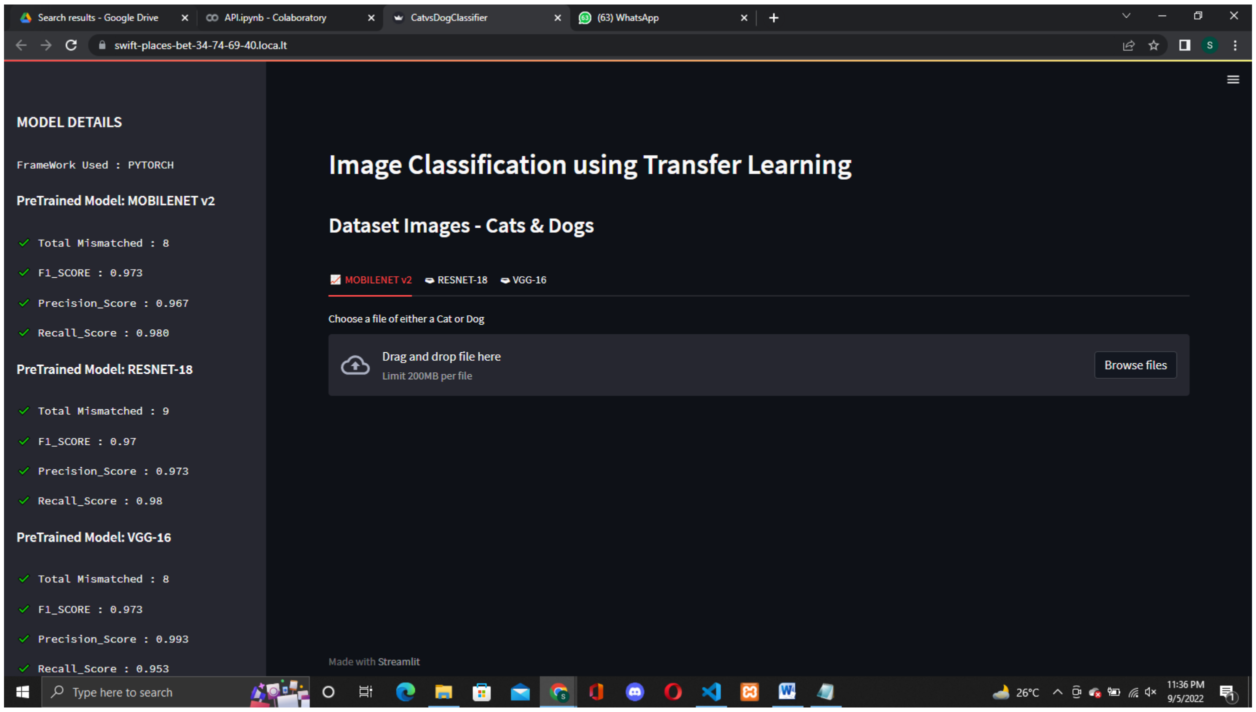
```
%%writefile /content/streamlit_app.py
from PIL import Image
import streamlit as st
import torch
import numpy as np
import time
from torchvision import transforms
import torch.nn.functional as F

model = torch.load("/content/drive/MyDrive/data/mobilenet.pt")
model1 = torch.load("/content/drive/MyDrive/data/resnet18.pt")
model2 = torch.load("/content/drive/MyDrive/data/VGG.pt")

st.set_page_config(
     page_title="CatvsDogClassifier",
     layout="wide",  )

with st.sidebar:
 st.header("MODEL DETAILS", )
 st.text(" FrameWork Used : PYTORCH")
 st.subheader("PreTrained Model: MOBILENET_V2")
 st.text("✔ Total Mismatched : 8")
 st.text("✔ F1_SCORE : 0.973")
 st.text("✔ Memory Size : 14 MB")
 st.subheader("PreTrained Model: RESNET-18")
 st.text("✔ Total Mismatched : 9")
 st.text("✔ F1_SCORE : 0.97")
 st.text("✔ Memory Size : 45 MB")
 st.subheader("PreTrained Model: VGG-16")
 st.text("✔ Total Mismatched : 8")
 st.text("✔ F1_SCORE : 0.973")
 st.text("✔ Memory Size : 528 MB")

st.header("Image Classification using Transfer Learning", )
st.subheader("Dataset Images - Cats & Dogs")
tab1, tab2, tab3= st.tabs(["□  MOBILENET ", "□   RESNET-18", "□   VGG16"])

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
uploaded_file = tab1.file_uploader("Choose a file of either a Cat or Dog")
uploaded_file2 = tab2.file_uploader("Choose a Image of either a Cat or Dog")
uploaded_file3 = tab3.file_uploader("Choose a Picture of either a Cat or Dog")

mean = (0.485, 0.456, 0.406)
std = (0.229, 0.224, 0.225)
```

```python
transf= transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(mean, std)
    ])

if uploaded_file is not None:
  image = Image.open(uploaded_file)
  since = time.time()
  img = transf(image)
  img = img.unsqueeze(0)
  img = img.to(device)
  pred=model(img)
  pred1 = F.softmax(pred, dim=1)[:, 1]
  pred2 = pred1[0] > 0.55
  col1, col2 = tab1.columns(2)
  with col1:
   if pred2==1:
     col1.success(" □ □ □  PREDICTION = CAT  □ □ □ ")
     col1.text('=>   Probability : {:4f}'.format(pred1.item()))
   else:
     col1.success(" □ □ □  PREDICTION = DOG  □ □ □")
     col1.text('=>   Probability : {:4f}'.format(1 - pred1.item()))
   time_elapsed = time.time() - since
   col1.text('=>   Inference Time: {:4f}'.format(
        time_elapsed))
  with col2:
   col2.image(image, width = 280)

if uploaded_file2 is not None:
  image = Image.open(uploaded_file3)
  since = time.time()
  img = transf(image)
  img = img.unsqueeze(0)
  img = img.to(device)
  pred=model1(img)
  pred1 = F.softmax(pred, dim=1)[:, 1]
  pred2 = pred1[0] > 0.55
  col1, col2 = tab2.columns(2)
  with col1:
   if pred2==1:
     col1.success(" □ □ □  PREDICTION = CAT  □ □ □ ")
     col1.text('=>   Probability : {:4f}'.format(pred1.item()))
   else:
     col1.success(" □ □ □  PREDICTION = DOG  □ □ □")
     col1.text('=>   Probability : {:4f}'.format(1 - pred1.item()))
   time_elapsed = time.time() - since
   col1.text('=>   Inference Time: {:4f}'.format(
        time_elapsed))
  with col2:
   col2.image(image, width = 280)
```

```
if uploaded_file3 is not None:

  image = Image.open(uploaded_file3)
  since = time.time()
  img = transf(image)
  img = img.unsqueeze(0)
  img = img.to(device)
  pred=model2(img)
  pred1 = F.softmax(pred, dim=1)[:, 1]
  pred2 = pred1[0] > 0.55
  col1, col2 = tab3.columns(2)
  with col1:
   if pred2==1:
     col1.success(" □ □ □  PREDICTION = CAT  □ □ □ ")
     col1.text('=>   Probability : {:4f}'.format(pred1.item()))
   else:
     col1.success(" □ □ □  PREDICTION = DOG  □ □ □")
     col1.text('=>   Probability : {:4f}'.format(1 - pred1.item()))
   time_elapsed = time.time() - since
   col1.text('=>   Inference Time: {:4f}'.format(
        time_elapsed))
  with col2:
   col2.image(image, width = 280)
```

6.3 Start Hosting API

```
!streamlit run '/content/streamlit_app.py' & npx localtunnel --port 8501
```


Search results - Google Drive
API.ipynb - Colaboratory
CatvsDogClassifier
(63) WhatsApp
swift-places-bet-34-74-69-40.loca.lt

MODEL DETAILS

FrameWork Used : PYTORCH

PreTrained Model: MOBILENET v2

✓ Total Mismatched : 8

✓ F1_SCORE : 0.973

✓ Precision_Score : 0.967

✓ Recall_Score : 0.980

PreTrained Model: RESNET-18

✓ Total Mismatched : 9

✓ F1_SCORE : 0.97

✓ Precision_Score : 0.973

✓ Recall_Score : 0.98

PreTrained Model: VGG-16

✓ Total Mismatched : 8

✓ F1_SCORE : 0.973

✓ Precision_Score : 0.993

✓ Recall_Score : 0.953

Image Classification using Transfer Learning

Dataset Images - Cats & Dogs

Choose a file of either a Cat or Dog