PROJECT 2

Arizona State University

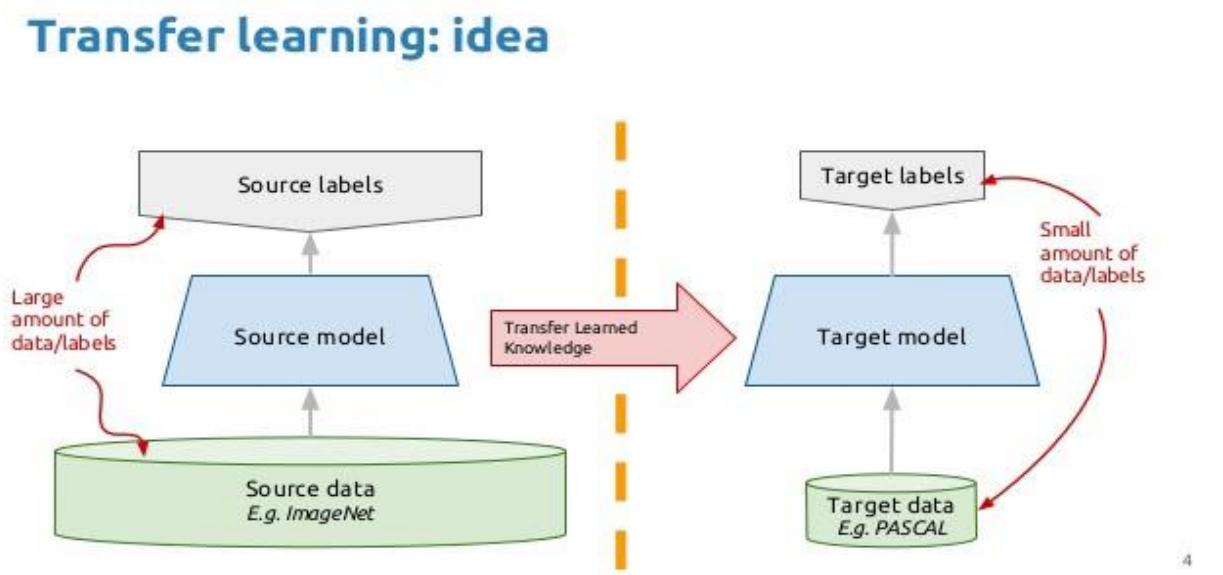
COURSE EEE 508

DIGITAL IMAGE AND VIDEO COMPRESSION

Transfer Learning in PyTorch

Aalap Doshi, Poojan Patel
(Group 6)

OBJECTIVE: - As main objective of the project we have to perform transfer learning for the pretrained datasets assigned to our group. For the Transfer learning we have used the Fine-Tuning method. Once model is trained, we were assigned to perform Fine-tuning on the pre-trained model to adapt to our new data sets. The next task of the project is Feature Extraction and Fine Tuning with data augmentation. We have used the inception model to perform the Transfer Learning.



Main Elements of the Project: -

- Pytorch
- Inception Model
- Dataset – ImageNet (Places)
- Fine-tuning
- Feature Extraction

Let's consider each element individually

1) **Pytorch :-**
   - There are two major ways of transfer learning in the Pytorch:-
     1) Finetuning the convnet
     2) Convnet as Fixed Feature extractor

   As per the method of fine tuning, instead of using the random initialization, we can use the pre-trained network to initialize our network. Here, we have used the network of ImageNet.

Pytorch is basically a replacement for a numpy to run deep learning on the GPU. It is also a smart choice for its flexibility and speed. Pytorch contains the AUTOGRAD function which provides automatic differentiation packages for all operations in tensors

Pytorch provides inbuild libraries for the Transfer Learning first we need to load that libraries as described below.

```
from __future__ import print_function, division

import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
import numpy as np
import torchvision
from torchvision import datasets, models, transforms
import matplotlib.pyplot as plt
import time
import copy
```

To load the data we will require torch vision and torch.utils.data packages

We will require the general function to train the model which is train_model().

- Neural Networks in the Pytorch:-

Neural Networks can be computed using the torch.nn. to define the model and differentiate the output nn.module contains layers , and method forward (input) that returns the output.

t is a simple feed-forward network. It takes the input, feeds it through several layers one after the other, and then finally gives the output.

A typical training procedure for a neural network is as follows:

- Define the neural network that has some learnable parameters (or weights)
- Iterate over a dataset of inputs
- Process input through the network
- Compute the loss (how far is the output from being correct)
- Propagate gradients back into the network's parameters
- Update the weights of the network, typically using a simple update rule: weight = weight - learning rate * gradient

There are several types of models which works on Neural Networks the as defined below

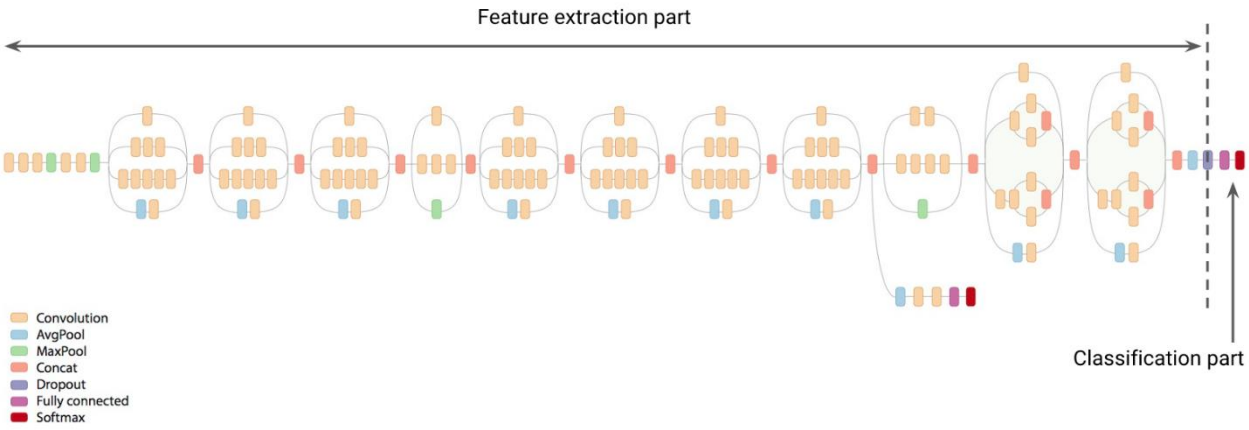
1) Resnet

2) Alex net
3) VGG
4) Squeeze net
5) Dense net
6) Inception

And we have worked on the Inception model as assigned to our group.

2) Inception Model: -

- The pre-trained Inception-v3 model achieves state-of-the-art accuracy for recognizing general objects with 1000 classes, like "Places", "Animal", and "Households". The model extracts general features from input images in the first part and classifies them based on those features in the second part.



In transfer learning, when you build a new model to classify your original dataset, you reuse the feature extraction part and re-train the classification part with your dataset. Since you don't have to train the feature extraction part (which is the most complex part of the model), you can train the model with less computational resources and training time.

This network is unique because it has two output layers when training. 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. Note, when testing we only consider the primary output.

```
model_name == "inception":
 """ Inception v3
 Be careful, expects (299,299) sized images and has auxiliary output
 """
 model_ft = models.inception_v3(pretrained=use_pretrained)
 set_parameter_requires_grad(model_ft, feature_extract)
 # Handle the auxilary net
 num_ftrs = model_ft.AuxLogits.fc.in_features
 model_ft.AuxLogits.fc = nn.Linear(num_ftrs, num_classes)
 # Handle the primary net
 num_ftrs = model_ft.fc.in_features
 model_ft.fc = nn.Linear(num_ftrs,num_classes)
 input_size = 299
```

Above code is used for the initialization and validation of the inception model in pytorch. As Inception model works on the concept of Computational neural network. Inception v3 is the version of the inception model we have used in this project.

- **Inception Net v3** incorporated many upgrades stated for Inception v2, and in addition used the following:

    1. RMSProp Optimizer.
    2. Factorized 7x7 convolutions.
    3. BatchNorm in the Auxillary Classifiers.
    4. Label Smoothing (A type of regularizing component added to the loss formula that prevents the network from becoming too confident about a class. Prevents over fitting).

3) Datasets: -

As Datasets are one of the key features for the transfer learning. As provided one of these datasets to work on. The datasets are subsets of existing datasets. Figure shows example images from these datasets. We required to fine-tune models that were originally trained on the ImageNet dataset. With transfer learning we alleviate some of the problems with using small datasets. Typically, if we tried to train a network from scratch on a small dataset, we might experience overfitting problems. But in transfer learning, we start with some network trained on a much larger dataset. Because of this, the features from the pre-trained network are not likely to overfit our data, yet still likely to be useful for classification.

Dataset	Description	Categories	Train	Test
Animals	iNat2017 challenge	9	1384	466
Places	Places Dataset	9	1437	367
Household	iMaterialist 2018 challenge	9	1383	375
Caltech101	CVPR 2004 Workshop	30	1500	450

4) Finetuning: -

We have preferred two methods for Finetuning: -

1) Finetuning with data augmentation
2) Finetuning without data augmentation

1)Fine Tune without Data augmentation: -

In **finetuning**, we start with a pretrained model and update *all* of the model's parameters for our new task, in essence retraining the whole model. We have used Inception model, Inception v3 is a widely-used image recognition model that has been shown to attain greater than 78.1% accuracy on the ImageNet dataset. The model is the culmination of many ideas developed by multiple researchers over the years. It is based on the original paper: "Rethinking the Inception Architecture for Computer Vision" by Szegedy.

The model itself is made up of symmetric and asymmetric building blocks, including convolutions, average pooling, max pooling, concats, dropouts, and fully connected layers. Batch norm is used extensively throughout the model and applied to activation inputs. Loss is computed via Softmax.

type	patch size/stride or remarks	input size
conv	3×3/2	299×299×3
conv	3×3/1	149×149×32
conv padded	3×3/1	147×147×32
pool	3×3/2	147×147×64
conv	3×3/1	73×73×64
conv	3×3/2	71×71×80
conv	3×3/1	35×35×192
3×Inception	As in figure 5	35×35×288
5×Inception	As in figure 6	17×17×768
2×Inception	As in figure 7	8×8×1280
pool	8 × 8	8 × 8 × 2048
linear	logits	1 × 1 × 2048
softmax	classifier	1 × 1 × 1000

We have used Google Colab's CPU to run our python code. **Google Colab** is a free cloud service to develop deep learning applications using popular libraries such as Keras, TensorFlow, PyTorch, and OpenCV. To run google colab we connect our drive with Google Colab and then create a python file. All the three codes are under My Drive/Project DIVP and also all the saved models are under same path.

Step By Step Explanation of the code for Finetuning without DataAugmentation:

1. We mount our drive so that the python program can access all the dataset from the drive.

2. We import torchvision and for our code we are using Pytorch as library.

3. Check whether the environment as GPU or not.

4. We have defined the train_model function, it handles the training and validation of a given model. As input, it takes a PyTorch model, a dictionary of dataloaders, a loss function, an optimizer, a specified number of epochs to train and validate(in this code we have written '**val' as 'test')** for, and a boolean flag for when the model is an Inception model. The *is_inception* flag is used to accomodate the *Inception v3* model, as that architecture uses an auxiliary output and the overall model loss respects both the auxiliary output and the final output. The function trains for the specified number of epochs and after each epoch runs a full validation step. It also keeps track of the best performing model (in terms of validation accuracy), and at the end of training returns the best performing model. After each epoch, the training and validation accuracies are printed.

5. For finetuning we have to update all the model parameters for our new task so we keep .requires_grad=True so that it computes the gradients of all the layers of the model.

6. Then we Initialize and Reshape the model. For Inception_v3 the required input size is (299,299). It has two output layers when training. 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. Note, when testing we only consider the primary output.

7. We reshape both the layers when we are finetuning this model. We have 9 classes.

```
model.AuxLogits.fc = nn.Linear(768, 9)
model.fc = nn.Linear(2048, 9)
```

8. Then we load the data and in the transforms we do not perform augmentation.

9. We define the optimization algorithm for finetuning

10. Then we train the model and check the train accuracy and validation accuracy as each epoch runs.

11. For finetuning we work on the pretrained model and update all the model parameters so the validation accuracy for the finetuned network is between 80-87% but the original accuracy for the inception_v3 model is around 78%.

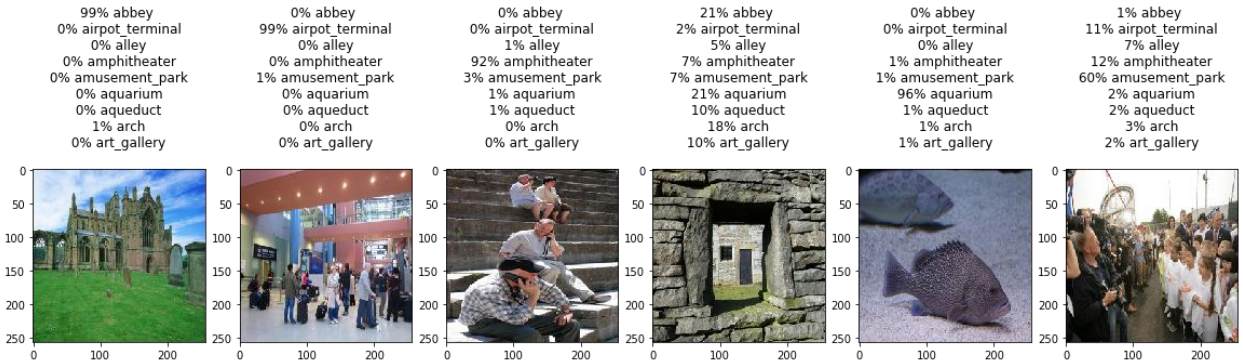
12. After training the model we save the model named “finetune.pt” and then we test the model giving random image for the test dataset as input to the test algorithm.

13. This will predict the percentage of the accuracy for each classes and usually the class from which the image is taken has the highest number of percentage.

14. Then we plot the confusion matrix.

For fine tuning the **Best Test(Validation) Accuracy is 82.56% and Train Accuracy is 98.9%**

Tested on Random Image:-



2)Finetune With Data Augmentation: -

Finetune with Data Augmentation follows the same steps as above for finetune without data augmentation the only change is that for this method we augment the data and increase the dataset so that the model can be trained on more data.

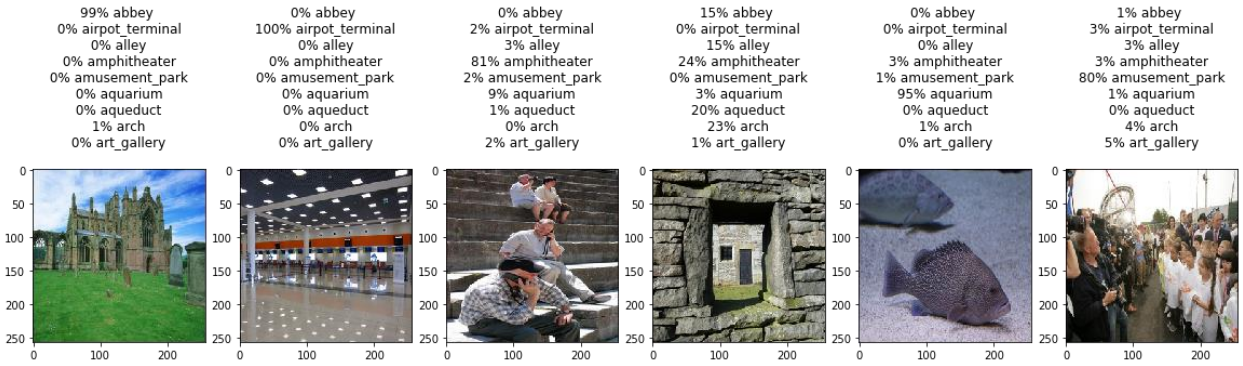
We are using the "Places" dataset and use inception model. Data Augmentation in Pytorch is performed by the "transforms" operations. This are applied to your original images at every batch generation. So, your dataset is left unchanged, only the batch images are copied and transformed every iteration. The "transforms" are used both for data preparation (resizing/cropping to expected dimensions, normalizing values, etc.) and for data augmentation (randomizing the resizing/cropping, randomly flipping the images, etc.)

The accuracy of the augmented data is less than the one without data augmentation because we are using places as the dataset and on performing horizontal and vertical flips and such types of augmentation techniques will hamper the train data because will cause the model to learn the new image generated per batch as a new separate image and that will hamper the validation accuracy. Generally, with augmentation the output of the validation accuracy should be more but in our case it is different. It doesn't drop by much percentage but drops by some percentage.

For this code we have used Adam as our optimizer.

For fineplus the **Best Test(Validation) Accuracy is 77.80% and Train Accuracy is 98%**

Tested on Random Image:



5)Feature Extraction: -

Step by Step Explanation of the code for Feature Extraction:

1. We mount our drive so that the python program can access all the dataset from the drive.
2. We import torch vision and for our code we are using Pytorch as library.

3. Check whether the environment as GPU or not.

4. We have defined the train_model function, it handles the training and validation of a given model. As input, it takes a PyTorch model, a dictionary of data loaders, a loss function, an optimizer, a specified number of epochs to train and validate (in this code we have written '**val' as 'test')** for, and a Boolean flag for when the model is an Inception model. The *is inception* flag is used to accommodate the *Inception v3* model, as that architecture uses an auxiliary output and the overall model loss respects both the auxiliary output and the final output. The function trains for the specified number of epochs and after each epoch runs a full validation step. It also keeps track of the best performing model (in terms of validation accuracy), and at the end of training returns the best performing model. After each epoch, the training and validation accuracies are printed.

5. **For feature extraction**, we start with a pretrained model and only update the final layer weights from which we derive predictions. It is called feature extraction because we use the pretrained CNN as a fixed feature-extractor, and only change the output layer. We are feature extracting and only want to compute gradients for the newly initialized layer then we want all of the other parameters to not require gradients.

6. When feature extracting, we only want to update the parameters of the last layer, or in other words, we only want to update the parameters for the layer(s) we are reshaping. Therefore, we do not need to compute the gradients of the parameters that we are not changing, so for efficiency we set the. requires_grad attribute to False. This is important because by default, this attribute is set to True. Then, when we initialize the new layer and by default the new parameters have. requires_grad=True so only the new layer's parameters will be updated

7. Then we Initialize and Reshape the model. For Inception_v3 the required input size is (299,299). It has two output layers when training. 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. Note, when testing we only consider the primary output.

8. Then we load the data and in the transforms. .

9. We define the optimization algorithm for feature extracting. If feature_extract=True we manually set all of the parameters. requires_grad attributes to False. Then the reinitialized layer's parameters have. requires_grad=True by default. So now we know that all parameters that have. requires_grad=True should be optimized.

10. Then we train the model and check the train accuracy and validation accuracy as each epoch runs.

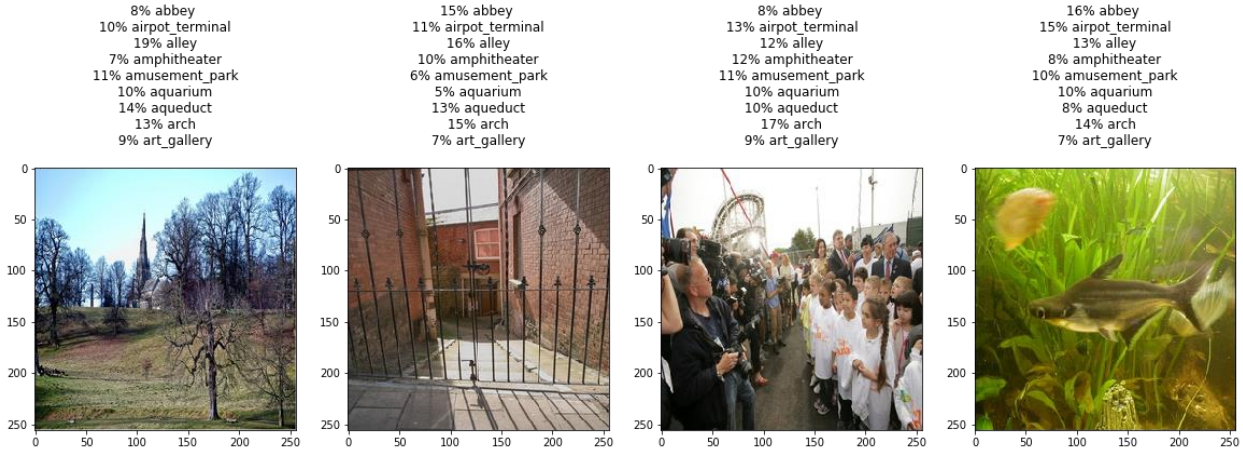
11. After training the model we save the model named "feature.pt" and then we test the model giving random image for the test dataset as input to the test algorithm.

12. This will predict the percentage of the accuracy for each classes and usually the class from which the image is taken has the highest number of percentage.

13. Then we plot the confusion matrix.

For feature extraction we trained the model first and we were getting **75%** accuracy but due to some problem now we are getting only **15%** accuracy. We trained the model many times after that and also made a new notebook made a whole new code but the accuracy is not increasing. However, we have understood the concepts of feature extraction. We feel that there must be some problem with overfitting of the data in the model but we are not sure about this error.

Tested on Random Image:



How to Test the model:-

- All the code have been performed on google colab so the paths for the train dataset and test dataset are accordingly. We can come in person to test the model in our google colab.
- To test the model we save the model and then perform testing. In short we load our model and then take some images from the “test dataset” and check whether the accuracy of the prediction is true or not.
- In our code to test the model we reset all the cells again and then again mount the drive of the google drive.
- Then we run all the cells except the Train and Evaluate cell and the torch.save() cell. Because we don’t train the model everytime we want to test the model.
- Model=torch.load() will load the model from the path set and then we can test the model with different images.

Tasks:

All the tasks are been performed by both Aalap Doshi and Poojan Patel. Aalap Doshi has worked on Finetune and Fineplus code and Poojan Patel has worked on Feature Extraction code, however both have equal inputs in all the three codes and worked on it together. Poojan Patel came up with the solution for confusion matrix in all the three codes. Aalap Doshi came up with the solution for saving and loading the model on drive as the coding is on google colab. In conclusion, both the team members have performed equally on the project.

Conclusion: -

The inception_v3 is a widely-used image recognition model that has been shown to attain greater than 78.1% accuracy on the ImageNet dataset. Throughout our work, Inception has shown us that
(a) It gives a good accuracy when it comes to transfer learning with and without augmentation of the images.
(b) it trains faster on a low processing power better than a simple CNN. So, we can conclude that the promises made by Inception are held well when tested.

However, this is not the final definitive conclusion, and there are many further ways we can move from this point. Considering the limitations, we had, the Inception has not been critically analyzed from ground up. Overcoming the limitations would result into better understanding the pros and cons of such architecture. Nonetheless, so far, the results are promising enough to open doors to many aspects of practical applications. Inception can be the best architecture to be implemented into the devices with low processing units, until a better model succeeds the results Inception has shown.

References:-

1) https://core.ac.uk/download/pdf/74351939.pdf
2) https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html
3) https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html
4) https://github.com/pytorch/vision/blob/master/torchvision/models/inception.py
5) http://pytorch.org/tutorials/beginner/pytorch_with_examples.html
6) https://github.com/pytorch/vision/tree/master/torchvision/models

7) https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html
8) https://pytorch.org/docs/stable/nn.functional.html
9) https://stackoverflow.com/questions/50170011/adapting-pytorch-softmax-function
10) https://stackoverflow.com/questions/55120789/convolutional-auto-encoder-error-runtimeerror-input-type-torch-cuda-byteten
11) https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html