Udacity Project Report

Machine Learning Engineer Nanodegree Project-4

Dog Breed Classification

Efehan Danisman / 31.10.2020

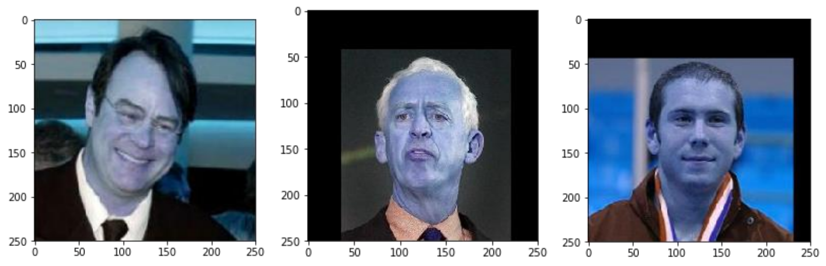
1. Introduction

1.1. Problem Statement

Image classification is a growing field with recent breakthroughs along with an increase of computing power and new methods developed in Convolutional Neural Networks such as Transfer Learning[1] or models that are developed with huge datasets. For example, ImageNet developed with more than 14 million images for object detection for more than 20.000 categories[2]. Breed classification is a problem that comes from this domain and aims to classify dog breeds. There was even a Kaggle competition about that![3] Certain models that can be used as pre-trained models are behind this breakthrough as state-of-the-art models such as ImageNet, VGG-16, ResNet50 and EfficientNet[4].

At this project, I developed a Convolutional Neural Network (CNN) to classify dog breeds according to their types if the application is fed with a dog photo. In case it is fed with a human photo, application returns the most resembling dog breed and in case of any other photo, it will return an error.

Dataset consists 13233 human images with 250x250 size and 8351 dog images with 836 of them is used for test, 836 of them is used for validation. This is a multi-class classification problem for dogs with 133 different classes each of them consists between 0.5% to 1% of the whole dataset. Below you can see some example images from human and dog datasets.



[1] https://www.tensorflow.org/tutorials/images/transfer_learning

[2] http://www.image-net.org/

[3] https://www.kaggle.com/c/dog-breed-identification/overview

[4] https://github.com/onnx/models#image_classification

Figure 1 - Sample Images from Human Dataset

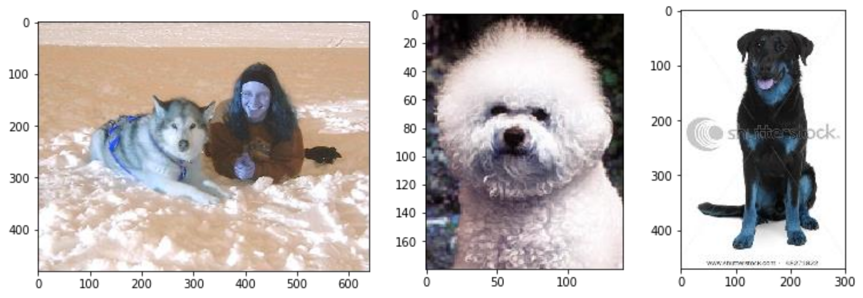


Figure-2 Sample Images from Dog Dataset

1.2. Objective

Since the project framework is developed by Udacity, benchmarks to pass are laid out in the project rubric. First, we need to create a human detector which will be classifying images as human or not. Then we need to create two convolutional neural networks, one from scratch with at least 10% accuracy and another one using state of the art methods with more than 60% accuracy. At the end, models will be merged in one small app and returns the dog breed if it is fed with a dog picture.

Since this is a multi-class classification problem, accuracy is a great metric to evaluate the model with ratio of the number of correct predictions to the total number of observations.

2. Methodology & Results

2.1. Human Face Detector

I started the project with detecting whether a picture contains human face or not. To classify images, OpenCV's[5] Cascade Classifier. Cascade Classifier is based on ancestor of the Xgboost classifier, Adaboost[6].

[5] https://docs.opencv.org/master/db/d28/tutorial_cascade_classifier.html
[6] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

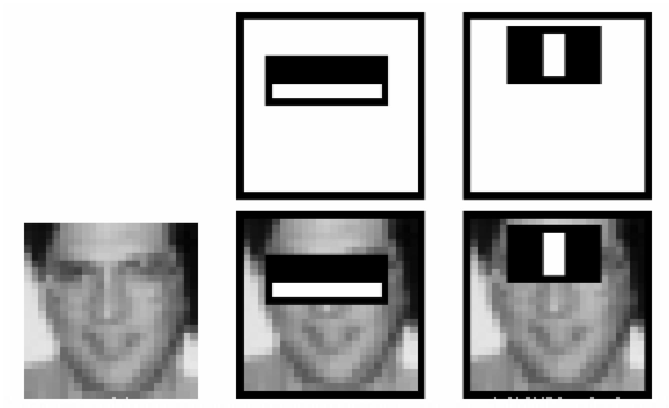
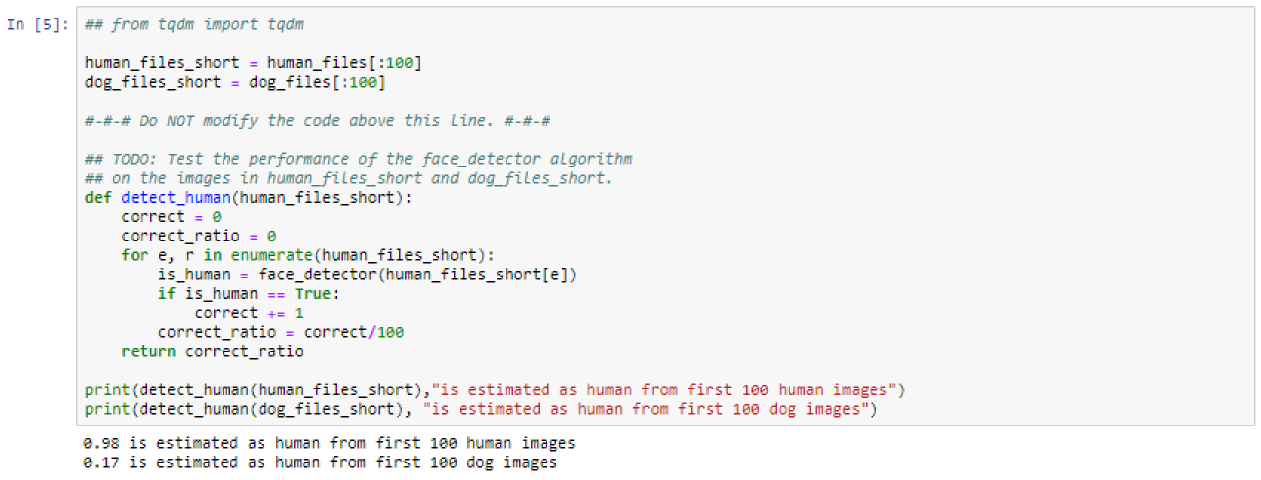


Figure 3- How Cascade Classifier Detect Relevant Features?

Reference: https://docs.opencv.org/master/db/d28/tutorial_cascade_classifier.html

As a result, 98% of the human images are recognized as human's while 17% of the dog images are recognized as humans. Due to computational reasons, I only reported first 100 pictures here.



```
In [5]: ## from tqdm import tqdm

        human_files_short = human_files[:100]
        dog_files_short = dog_files[:100]

        #-#-# Do NOT modify the code above this line. #-#-#

        ## TODO: Test the performance of the face_detector algorithm
        ## on the images in human_files_short and dog_files_short.
        def detect_human(human_files_short):
            correct = 0
            correct_ratio = 0
            for e, r in enumerate(human_files_short):
                is_human = face_detector(human_files_short[e])
                if is_human == True:
                    correct += 1
                correct_ratio = correct/100
            return correct_ratio

        print(detect_human(human_files_short),"is estimated as human from first 100 human images")
        print(detect_human(dog_files_short), "is estimated as human from first 100 dog images")

        0.98 is estimated as human from first 100 human images
        0.17 is estimated as human from first 100 dog images
```

Figure 4- Result of the Cascade Classifier

2.2. Dog Detector

For dog detector, VGG16 is recommended to identify dogs. It is pre-trained with weights from ImageNet which has more than 10 million images in more than 1000 categories[7]. VGG16 is developed by group of scientists from Oxford University in 2015 and reached top-5 test accuracy in Image-Net challenge[8]. Below you can see the VGG16 architecture.

[7] https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a

[8] http://www.image-net.org/challenges/LSVRC/2014/results

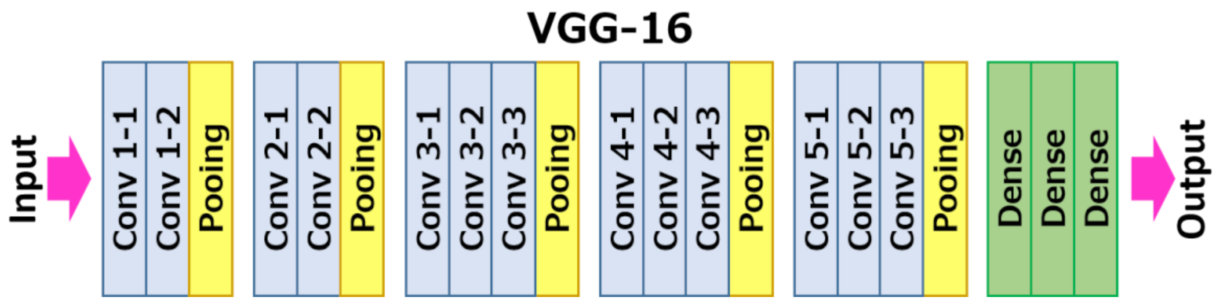


Figure 5- VGG16 Architecture

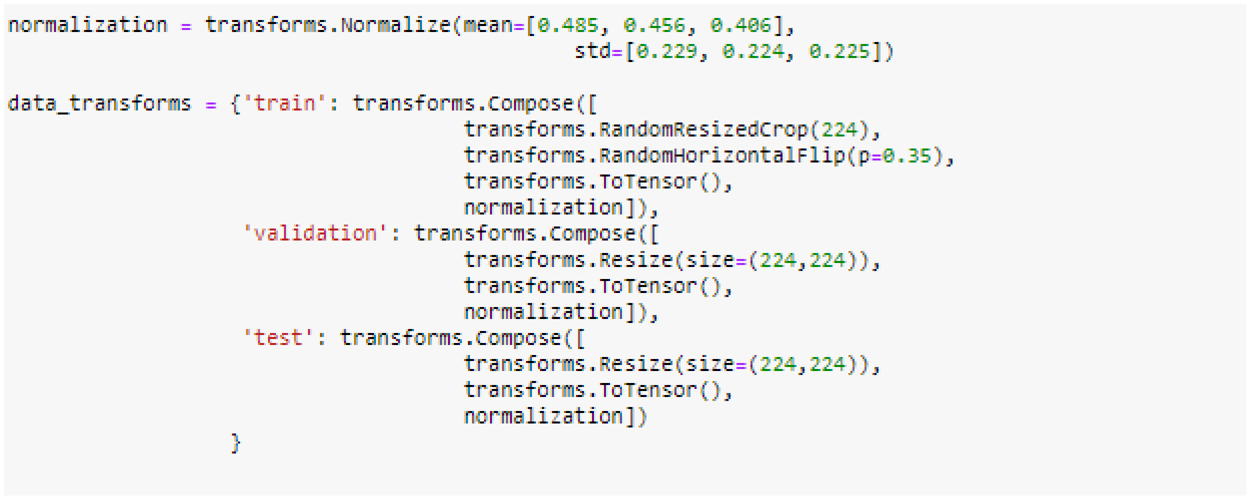
Reference: https://neurohive.io/en/popular-networks/vgg16/

VGG worked incredibly well, with 100% accuracy on classifying dogs and not classifying any of the humans as dog. Since it worked well, I decided not to continue the optional task to try other state of the art methods such as ResNet-50 and Inception-v3.

2.3. Creating a Dog Breed Classifier from Scratch

2.3.1. Data Transformation

Before creating a classifier, I pre-processed images using transforms class of PyTorch[9] so that model will generalize better. First randomly cropped the images into size of 224 pixels. Moreover, images are randomly flipped (35% of the images) so that model see different kind of images. Lastly, images are normalized with the values specified in VGG16 model. I did some other trial and errors such as affine transformation however model performed so poorly (around 1% accuracy) hence I did not report it.


```
normalization = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                     std=[0.229, 0.224, 0.225])

data_transforms = {'train': transforms.Compose([
                              transforms.RandomResizedCrop(224),
                              transforms.RandomHorizontalFlip(p=0.35),
                              transforms.ToTensor(),
                              normalization]),
                   'validation': transforms.Compose([
                              transforms.Resize(size=(224,224)),
                              transforms.ToTensor(),
                              normalization]),
                   'test': transforms.Compose([
                              transforms.Resize(size=(224,224)),
                              transforms.ToTensor(),
                              normalization])
                  }
```

[9] https://pytorch.org/docs/stable/torchvision/transforms.html

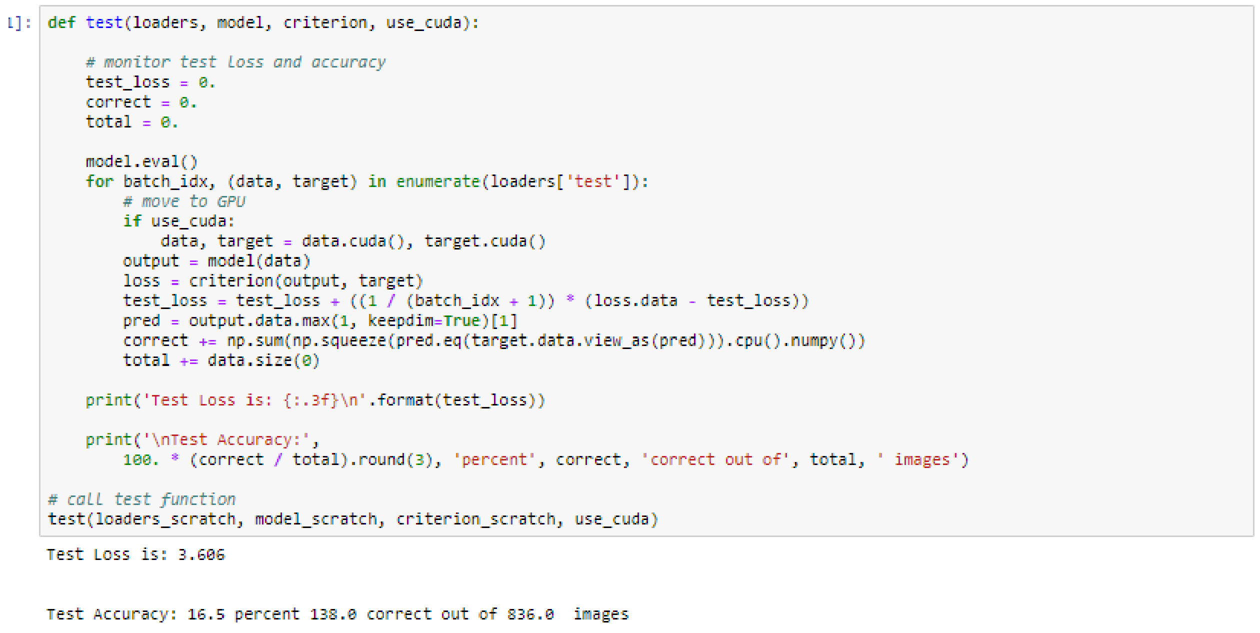
Figure 6 - Data Transforms

2.3.2. Convolutional Neural Network from Scratch

I developed a three layers network with batch normalization and 1 padding. In order to avoid overfitting, I did max pooling and used drop-out with 15% of the neurons. Below you can see the architecture of the model.

```
Net(
  (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3): Conv2d(32, 64, kernel_size=(2, 2), stride=(1, 1), padding=(1, 1))
  (conv3_bn): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=50176, out_features=500, bias=True)
  (fc1_bn): BatchNorm1d(500, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc2): Linear(in_features=500, out_features=133, bias=True)
  (dropout): Dropout(p=0.15)
)
```

As optimizer, I tried different state of the art optimizers such as Stochastic Gradient Descent, Adam and RMSPROP. For loss function, Cross-Entropy loss is a commonly used function. As a result, I reached the sufficient result with Adam optimizer, Cross-Entropy loss and learning rate of 0.001. For batch size, I selected 16 since with larger batches model performed poorly. After making couple of trials with number of epochs, I selected 20 epochs with reaching the best model at 19th. As a result of this model, I reached 16.5% accuracy which is overall poor however good enough for this task from scratch with small amount of data comparing to state-of-the-art pre-trained models.



```
1]: def test(loaders, model, criterion, use_cuda):

        # monitor test loss and accuracy
        test_loss = 0.
        correct = 0.
        total = 0.

        model.eval()
        for batch_idx, (data, target) in enumerate(loaders['test']):
            # move to GPU
            if use_cuda:
                data, target = data.cuda(), target.cuda()
            output = model(data)
            loss = criterion(output, target)
            test_loss = test_loss + ((1 / (batch_idx + 1)) * (loss.data - test_loss))
            pred = output.data.max(1, keepdim=True)[1]
            correct += np.sum(np.squeeze(pred.eq(target.data.view_as(pred))).cpu().numpy())
            total += data.size(0)

        print('Test Loss is: {:.3f}\n'.format(test_loss))

        print('\nTest Accuracy:',
            100. * (correct / total).round(3), 'percent', correct, 'correct out of', total, ' images')

    # call test function
    test(loaders_scratch, model_scratch, criterion_scratch, use_cuda)

    Test Loss is: 3.606


    Test Accuracy: 16.5 percent 138.0 correct out of 836.0  images
```

Figure 7 – CNN From Scratch Test Result

2.4. Creating a Dog Breed Classifier Using Transfer Learning

With transfer learning, accuracy bar is set much higher 60%. Since I reached top performance with current transformed version, I used same transformers for this model as well. Here I used linear transformation of 2048 inputs and selected Res-Net as my pre-trained model.

For optimizer and loss function, I continued same way with the previous project using Adam and Cross-Entropy loss with 20 epochs. I only increased the learning rate to 0.05

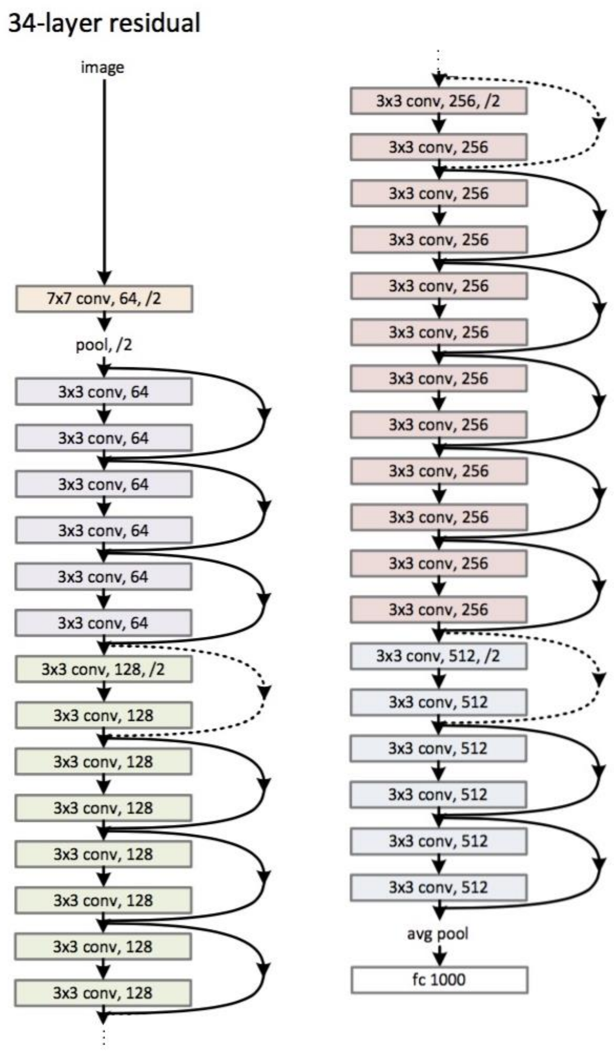
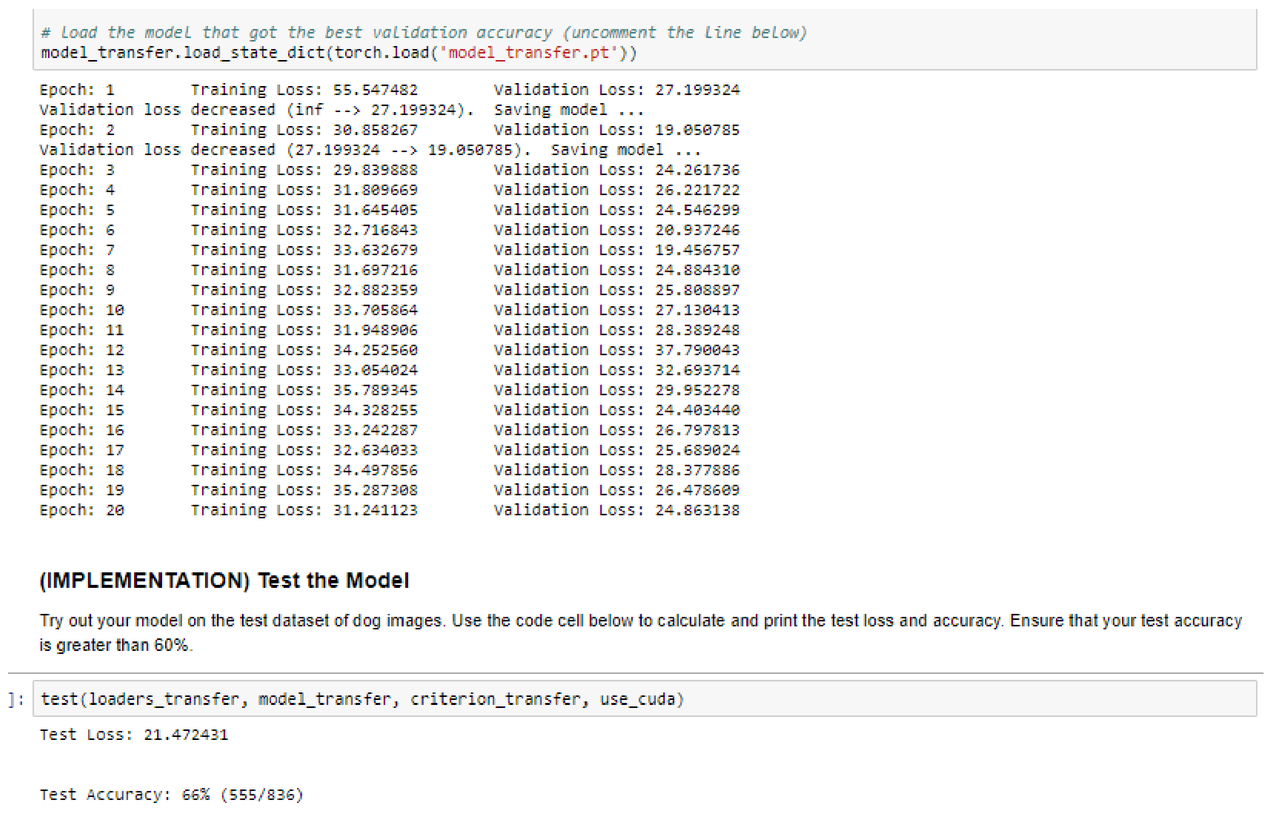


Figure 8 – Res-Net 50 Architecture

Reference: https://datascience.stackexchange.com/questions/33022/how-to-interpert-resnet50-layer-types/47489



```
# Load the model that got the best validation accuracy (uncomment the line below)
model_transfer.load_state_dict(torch.load('model_transfer.pt'))
```

```
Epoch: 1        Training Loss: 55.547482        Validation Loss: 27.199324
Validation loss decreased (inf --> 27.199324).  Saving model ...
Epoch: 2        Training Loss: 30.858267        Validation Loss: 19.050785
Validation loss decreased (27.199324 --> 19.050785).  Saving model ...
Epoch: 3        Training Loss: 29.839888        Validation Loss: 24.261736
Epoch: 4        Training Loss: 31.809669        Validation Loss: 26.221722
Epoch: 5        Training Loss: 31.645405        Validation Loss: 24.546299
Epoch: 6        Training Loss: 32.716843        Validation Loss: 20.937246
Epoch: 7        Training Loss: 33.632679        Validation Loss: 19.456757
Epoch: 8        Training Loss: 31.697216        Validation Loss: 24.884310
Epoch: 9        Training Loss: 32.882359        Validation Loss: 25.808897
Epoch: 10       Training Loss: 33.705864        Validation Loss: 27.130413
Epoch: 11       Training Loss: 31.948906        Validation Loss: 28.389248
Epoch: 12       Training Loss: 34.252560        Validation Loss: 37.790043
Epoch: 13       Training Loss: 33.054024        Validation Loss: 32.693714
Epoch: 14       Training Loss: 35.789345        Validation Loss: 29.952278
Epoch: 15       Training Loss: 34.328255        Validation Loss: 24.403440
Epoch: 16       Training Loss: 33.242287        Validation Loss: 26.797813
Epoch: 17       Training Loss: 32.634033        Validation Loss: 25.689024
Epoch: 18       Training Loss: 34.497856        Validation Loss: 28.377886
Epoch: 19       Training Loss: 35.287308        Validation Loss: 26.478609
Epoch: 20       Training Loss: 31.241123        Validation Loss: 24.863138
```

(IMPLEMENTATION) Test the Model

Try out your model on the test dataset of dog images. Use the code cell below to calculate and print the test loss and accuracy. Ensure that your test accuracy is greater than 60%.

```
]: test(loaders_transfer, model_transfer, criterion_transfer, use_cuda)
```

```
Test Loss: 21.472431


Test Accuracy: 66% (555/836)
```

Figure 9 – Result of the Transfer Learning Model

As a result, I reached 66% accuracy, which is better than the benchmark and completed the task.

3. Testing & Lessons Learnt

In order to test the model with different images, I uploaded some tricky images from my computer. First, I uploaded my cat's wild picture and model thought that it is a dog with breed of Great Pyrenees.

```
This is a dog and its breed isGreat pyrenees
```

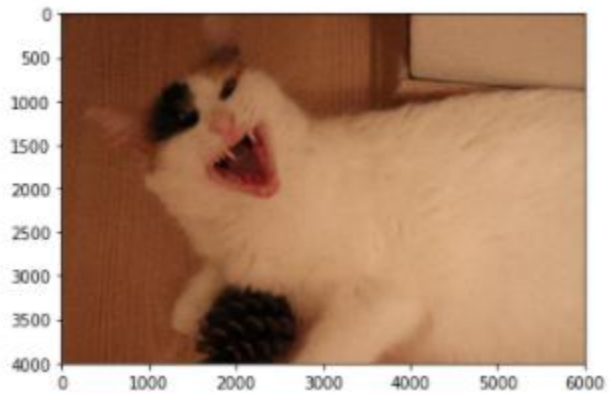


Figure 9 – Sample Cat Picture

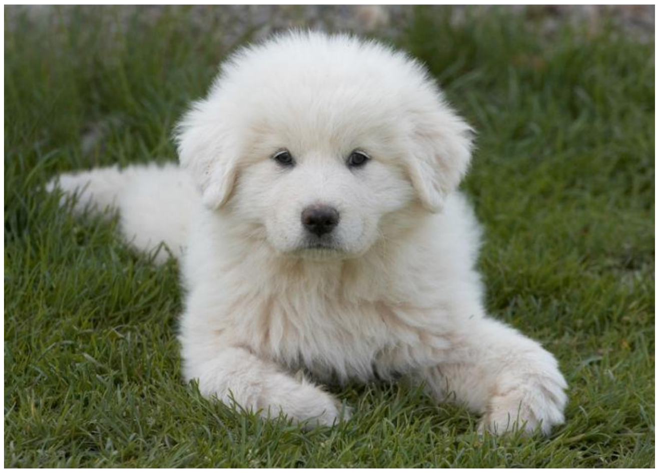


Figure 10 – Predicted Breed of Cat

After that I added a picture of myself and it is correctly classified as human with a resembling dog.

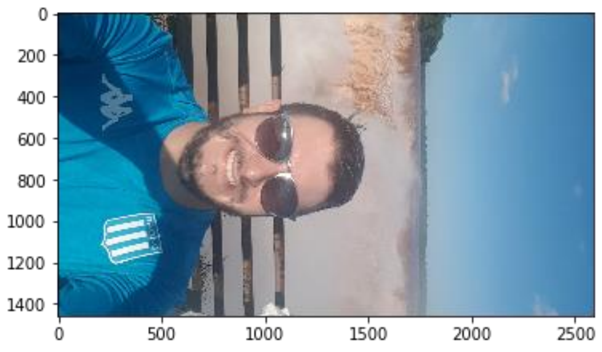


Figure 11 – My Picture to Test

Final model is not performing perfectly however fulfilling expectations in a multi-class problem classification with lots of classes. There are various ways to improve this accuracy such as increasing number of epochs, augmenting data to increase number of training samples and trying other state of the art methods.

4. References

1. Transfer Learning Tutorial, https://www.tensorflow.org/tutorials/images/transfer_learning
2. Image-Net architecture, http://www.image-net.org/
3. Kaggle Competititon on Dog Breed Identification, https://www.kaggle.com/c/dog-breed-identification/overview
4. Different image classification methods, https://github.com/onnx/models#image_classification
5. Cascade Classifier, https://docs.opencv.org/master/db/d28/tutorial_cascade_classifier.html
6. AdaBoost Classifier, https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html
7. ImageNet categories, https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a
8. VGG Architecture, http://www.image-net.org/challenges/LSVRC/2014/results
9. PyTorch Transformers, https://pytorch.org/docs/stable/torchvision/transforms.html