Machine Learning Engineer Nanodegree
Capstone Project

Juan Carlos Kuri Pinto

I. DEFINITION

Project Overview

The **Dog Breed Classification project,** or the **Dog App**, does 2 classifications. First, it classifies a photo into 3 categories: Human, dog, and other. Second, it classifies a photo into one of 133 dog breeds. Both classification problems are solved by using convolutional neural networks. And the human face detector is solved by using Haar cascades and boosting (Viola and Jones, 2001). In summary, the Dog App tells if a photo has a human, a dog, or another entity. Then it tells the dog breed that mostly resembles the entity in such photo.

The Dog App uses convolutional neural networks to classify photos of dogs, humans, and other animals into one of the 133 dog breeds that are contained in the dataset. Convolutional neural networks (LeCun and Bengio, 1995) are a special type of neural network that are translation-invariant and can find important invariant patterns in spatio-temporal signals. Convolutional neural networks are a powerful Deep Learning technique (Goodfellow, Bengio, and Courville, 2016). Deep Learning is a branch of Machine Learning (Mitchell, 1997) that uses neural networks to make machines learn salient and invariant patterns from data. Machine Learning is a branch of Artificial Intelligence (Russell and Norvig, 2020) whose algorithms are capable of learning from experience. And Artificial Intelligence is a branch of Computer Science whose intelligent algorithms try to make machine smarter.

To program this deep learning system with convolutional neural networks, 2 datasets are needed: The dataset of dogs and the dataset of humans.

The **dataset of dogs** was provided by Udacity and is already divided in 3 folders: train, valid, and test. The folder **train** has 133 subfolders (1 folder for each dog breed) that contain 6,680 images of dogs. The folder **valid** has 133 subfolders that contain 835 images of dogs. And the folder **test** has 133 subfolders that contain 836 images of dogs. In the dog project, the inputs are the photos and the outputs are the dog breeds. This dataset will serve to train the benchmark (a convnet trained from scratch) and the ResNet-50 with pretrained weights and transfer learning. This dataset will also serve to test the accuracy of the human face detector and the dog detector.

The **dataset of humans** was provided by Udacity and has 5,749 folders containing 13,233 images. In the dog project, it is used only for testing purposes, given that nothing is trained with this dataset. It is only useful to test the accuracy of the human face detector and the dog detector.

Problem Statement

Basically, we have 2 problems and their solutions. First, classifying a photo into 3 categories: Human, dog, and other. Second, classifying a photo into one of 133 dog breeds. Both classification problems are solved by using convolutional neural networks. And the human face detector is solved by using Haar cascades and boosting (Viola and Jones, 2001). The Dog App tells if a photo has a human, a dog, or another entity. Then it tells the dog breed that mostly resembles the entity in such photo.

The algorithm of Haar cascades and boosting uses Haar features to create weak classifiers capable of detecting human faces in groups of pixels. In each iteration of the cascade, weak classifiers do a vague job at classifying patterns. However, this vague job is compensated by the boosting algorithm, which is capable of creating a strong classifier out of weak classifiers like Haar features. How so? Each weak classifier do its lazy job and its mistakes are passed to the next level in the cascade. In this way, difficult-to-learn patterns are constantly retrained until the Haar features create a strong classifier.

Convolutional neural networks, or convnets, are a special type of neural network that are translation-invariant and can find important invariant patterns in spatio-temporal signals. Convnets have 2 types of layers: Convolutional layers and fully-connected layers. Convolutional layers are capable of finding translation-invariant patterns or features throughout the visual field. They represent visual patterns in a very efficient way because much less parameters are needed. Whereas, fully-connected layers have much more parameters that are location-specific, not location-invariant.

The problem of classifying photos into dog breeds and its solution using convnets are quantifiable, measurable, and replicable. The solution is quantifiable because the parameters or synapses that represent patterns in the convnet can be trained through the gradient descent technique applied to the loss function. Neurons are connected in a special neural architecture and generate outputs. Such outputs or predictions are compared to the ground-truth labels in order to compute a loss function, which is a metric of how adapted the convnet is. Hence, this solution is also measurable. Moreover, the solution is replicable because the metric is not only applicable to the patterns in the training dataset; but the metric is also applicable to the patterns in the validation dataset and in the test dataset. Once trained, the convnet can also give a probability distribution based on softmax that tells how similar a photo is to some particular dog breeds. And the photo does not need to have a dog in it. It could be a photo of a human or a photo of another animal. The convnet is capable of extrapolating patterns and making visual analogies, which is absolutely amazing.

Metrics

Both the loss and the accuracy are metrics to compare the benchmark model with the final solution, the ResNet-50 with pretrained weights and transfer learning.

Metric #1 – The Loss Function: The parameters or synapses that represent patterns in the convnet can be trained through the gradient descent technique applied to the loss function. Neurons are connected in a special neural architecture and generate outputs. Such outputs or predictions are compared to the ground-truth labels in order to compute a loss function or error function, which is a metric of how adapted the convnet is.

Metric #2 – Accuracy: Once the neural network has been trained with some examples, or through many epochs with the whole training dataset, we can test the predictions of the neural network against ground-truth labels. The percentage of correct predictions that agree with ground-truth labels is called accuracy. Accuracy is a metric that tells us how precise a machine learning algorithm is.

The benchmark model produced its best results at the epoch 23:

train_loss=0.005845, train_acc=89.31%
valid_loss=0.104890, valid_acc=14.97%

The best benchmark model was saved and produced the following results in the test dataset:

test_loss=0.120632, test_acc=15.07% (126/836)

The final solution, the ResNet-50 with pretrained weights and transfer learning, produced its best results at the epoch 24:

```
train_loss=0.000749, train_acc=99.19%
valid_loss=0.006908, valid_acc=87.54%
```

The best ResNet-50 model was saved and produced the following results in the test dataset:

test_loss=0.009719, test_acc=85.05% (711/836)

It is a really humbling experience to try to do your best at creating the best convnet possible with the best hyperparameters and yet the results pale in comparison to the results of the ResNet-50 with pretrained weights and transfer learning.

II. ANALYSIS

Data Exploration

To program this deep learning system with convolutional neural networks, 2 datasets are needed: The dataset of dogs and the dataset of humans.

The dataset of dogs was provided by Udacity and is already divided in 3 folders: train, valid, and test. The folder **train** has 133 subfolders (1 folder for each dog breed) that contain 6,680 images of dogs. The folder **valid** has 133 subfolders that contain 835 images of dogs. And the folder **test** has 133 subfolders that contain 836 images of dogs. In the dog project, the inputs are the photos and the outputs are the dog breeds. This dataset will serve to train the benchmark (a convnet trained from scratch) and the ResNet-50 with pretrained weights and transfer learning. This dataset will also serve to test the accuracy of the human face detector and the dog detector.

Dataset of Dogs:
https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/dogImages.zip

The dataset of humans was provided by Udacity and has 5,749 folders containing 13,233 images. In the dog project, it is used only for testing purposes, given that nothing is trained with this dataset. It is only useful to test the accuracy of the human face detector and the dog detector.

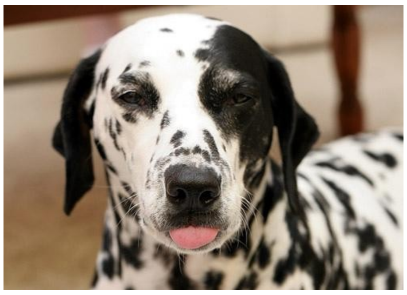
Dataset of Humans:
https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/lfw.zip

In the case of classifying dogs and humans, the labels are the dataset from which the photo was taken. In the case of classifying 133 dog breeds, the labels are the directory of dog breed from which the dog photo was taken.

Exploratory Visualization

The best way to explore and to visualize the dataset of dogs and the dataset of human is to watch some examples of photos.

Examples of dogs:



Dalmatian

French Bulldog

Borzoi

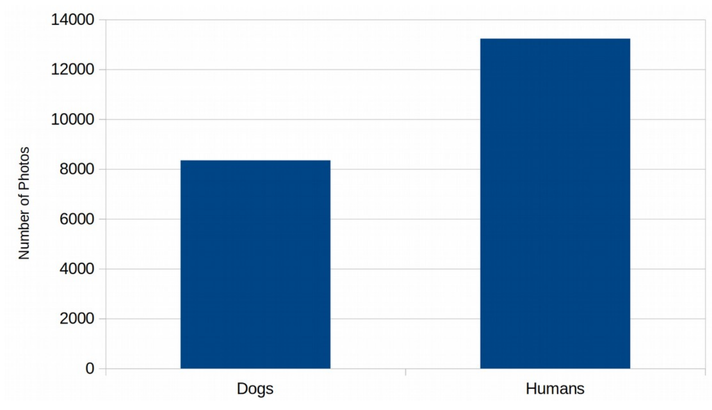
Examples of humans:

Alexandra Vodjanikova

Gustavo Noboa

Samira Makhmalbaf

The dataset of dogs has 8351 images and the dataset of 5749 humans has 13233 images:



Algorithms and Techniques

Basically, we have 2 problems to solve. First, classifying a photo into 3 categories: Human, dog, and other. Second, classifying a photo into one of 133 dog breeds. Both classification problems are solved by using convolutional neural networks (LeCun and Bengio, 1995). And the human face detector is solved by using Haar cascades and boosting (Viola and Jones, 2001). The Dog App tells if a photo has a human, a dog, or another entity. Then it tells the dog breed that mostly resembles the entity in such photo.

The algorithm of Haar cascades and boosting (Viola and Jones, 2001) uses Haar features to create weak classifiers capable of detecting human faces in groups of pixels. In each iteration of the cascade, weak classifiers do a vague job at classifying patterns. However, this vague job is compensated by the boosting algorithm, which is capable of creating a strong classifier out of weak classifiers like Haar features. How so? Each weak classifier do its lazy job and its mistakes are passed to the next level in the cascade. In this way, difficult-to-learn patterns are constantly retrained until the Haar features create a strong classifier.

Convolutional neural networks, or convnets, are a special type of neural network that are translation-invariant and can find important invariant patterns in spatio-temporal signals. Convnets have 2 types of layers: Convolutional layers and fully-connected layers. Convolutional layers are capable of finding translation-invariant patterns or features throughout the visual field. They represent visual patterns in a very efficient way because much less parameters are needed. Whereas, fully-connected layers have much more parameters that are location-specific, not location-invariant.

The parameters or synapses that represent patterns in the convnet can be trained through the gradient descent technique applied to the loss function. Neurons are connected in a special neural architecture and generate outputs. Such outputs or predictions are compared to the ground-truth labels in order to compute a loss function, which is a metric of how adapted the convnet is. Once trained, the convnet can also give a probability distribution based on softmax that tells how similar a photo is to some particular dog breeds. And the photo does not need to have a dog in it. It could be a photo of a human or a photo of another animal. The convnet is capable of extrapolating patterns and making visual analogies, which is absolutely amazing.

Benchmark

The benchmark model is a convnet trained from scratch, whose neural architecture is:

```
Net(
  (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
  (conv3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
  (conv4): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1))
  (drop): Dropout(p=0.25, inplace=False)
  (fc1): Linear(in_features=9216, out_features=2304, bias=True)
  (fc2): Linear(in_features=2304, out_features=576, bias=True)
  (fc3): Linear(in_features=576, out_features=133, bias=True)
)
```

And its forward method is:

```
    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))
        x = self.pool(F.relu(self.conv4(x)))
        x = x.view(x.size(0), -1)
        x = self.drop(F.relu(self.fc1(x)))
        x = self.drop(F.relu(self.fc2(x)))
        x = self.fc3(x)
        return x
```

Both the loss and the accuracy are metrics to compare the benchmark model with the final solution, the ResNet-50 with pretrained weights and transfer learning.

The benchmark model produced its best results at the epoch 23:

train_loss=0.005845, train_acc=89.31%
valid_loss=0.104890, valid_acc=14.97%

The best benchmark model was saved and produced the following results in the test dataset:
test_loss=0.120632, test_acc=15.07% (126/836)

III. METHODOLOGY

Data Preprocessing

The dataset of human images and the dataset of dog images have no abnormalities. And the only characteristics that needed to be addressed are the differences in image sizes. I don't use feature selection because convnets automatically create abstract features in neural space. Images are automatically transformed into patterns of feature space.

Here is the code for preprocessing the dataset of dog images:

```
image_size = 224
normalize = transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))

train_tranforms =\
 [transforms.Resize((image_size, image_size)),
  transforms.RandomHorizontalFlip(),
  transforms.ToTensor(),
  normalize]
train_transform = transforms.Compose(train_tranforms)

test_tranforms =\
 [transforms.Resize((image_size, image_size)),
  transforms.ToTensor(),
  normalize]
test_transform = transforms.Compose(test_tranforms)
```

In brief, the transformations for preprocessing the dataset of dog images are:

- resize to (224, 224) pixels by stretching;
- random horizontal flip;
- image to tensor;
- and normalization of tensors.

I only resize by stretching. I don't crop. I decided the size of the input tensor to be (224, 224) pixels. Why? Because it is the standard input size of ResNet-50. So, I use the same tranforms for both neural networks: The scratch neural network and the transfer neural network.

I augment the dataset only through horizontal flips. I don't use vertical flips because such transformation is unnatural. I don't use translations because convnets are already translation-invariant. I don't use rotations because rotations distort images.

Implementation

The Dog Breed Classification project has the following steps:

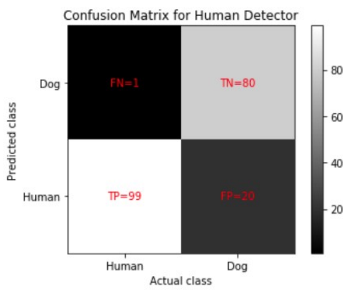
1. Importing the Human Dataset.
2. Importing the Dog Dataset.
3. Testing the human face detector.
4. Testing the dog detector.
5. Benchmark CNN (from scratch).
6. ResNet-50 (transfer learning).
7. Creating and testing the Dog App.

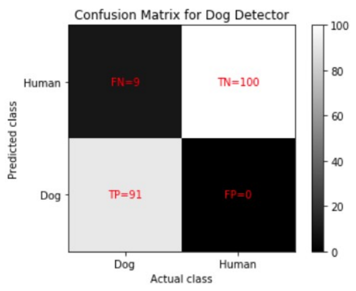
Part 1. Importing the Human Dataset.
Part 2. Importing the Dog Dataset.

In these 2 parts, the human dataset and the dog dataset are downloaded only if they don't exist in a local directory. Then, both datasets are loaded into memory as lists of images or dataloaders in PyTorch. Dataloaders apply the transformations described in the section Data Preprocessing.

Part 3. Testing the human face detector.
Part 4. Testing the dog detector.

In these 2 parts, the human face detector and the dog detector are tested with the 100 first samples of the human dataset and the dog dataset. Confusion matrices and accuracies are analyzed.





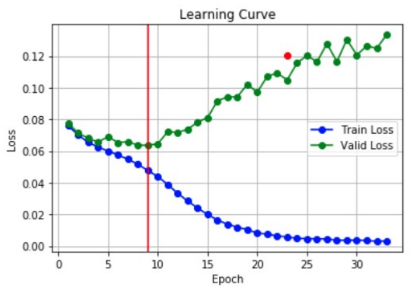
Human Detector	Dog Detector

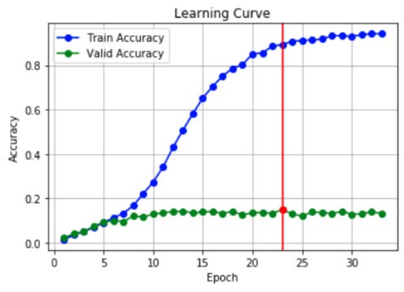
`P=100 (humans)`
`N=100 (dogs)`
 	
`TP=99`
`FP=20`
`TN=80`
`FN=1`
 	
`recall or true positive rate (TPR)=0.99`
`specificity or true negative rate (TNR)=0.8`
`precision or positive predictive value (PPV)=0.8319327731092437`
`negative predictive value (NPV)=0.9876543209876543`
 	
`acc=0.895`
`F1-score=0.904109589041096`
	`P=100 (dogs)`
	`N=100 (humans)`
	
	`TP=91`
	`FP=0`
	`TN=100`
	`FN=9`
	
	`sensitivity or true positive rate (TPR)=0.91`
	`specificity or true negative rate (TNR)=1.0`
	`precision or positive predictive value (PPV)=1.0`
	`negative predictive value (NPV)=0.9174311926605505`
	
	`acc=0.955`
	`F1-score=0.9528795811518325`

We can conclude that the Dog Detector based on convnets is much better than the Human Detector based on Haar boosted cascades. Why? Because Haar boosted cascades have issues with false positives. It's a well known problem.

Part 5. Benchmark CNN (from scratch).

The Benchmark CNN to classify dog breeds is created, trained from scratch, and tested. The learning curves of loss and accuracy are analyzed.





The benchmark model produced its best results at the epoch 23:

train_loss=0.005845, train_acc=89.31%
valid_loss=0.104890, valid_acc=14.97%

The best benchmark model was saved and produced the following results in the test dataset:

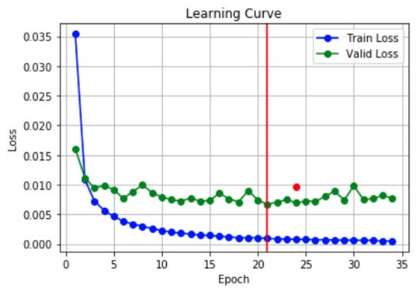
test_loss=0.120632, test_acc=15.07% (126/836)

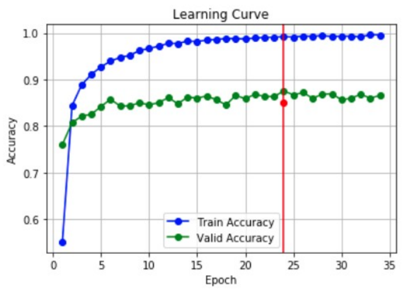
It was a little bit hard to consistently obtain a test accuracy greater than 10% in the Benchmark CNN. I played with many hyperparameters for the convolutional layers and the fully-connected layers. My test accuracy was between 8% and 11%. And it varied randomly, depending on the random numbers of specific runs. Then, I found the sweet spot after many experiments and I got a **test_acc=15.07%**

It is a really humbling experience to try to do your best at creating the best convnet possible with the best hyperparameters and yet the results pale in comparison to the results of the ResNet-50 with pretrained weights and transfer learning.

Part 6. ResNet-50 (transfer learning).

The ResNet-50 to classify dog breeds is created, trained, and tested. Training is done by using pretrained features and transfer learning. The learning curves of loss and accuracy are analyzed.





The final solution, the ResNet-50 with pretrained weights and transfer learning, produced its best results at the epoch 24:

train_loss=0.000749, train_acc=99.19%
valid_loss=0.006908, valid_acc=87.54%

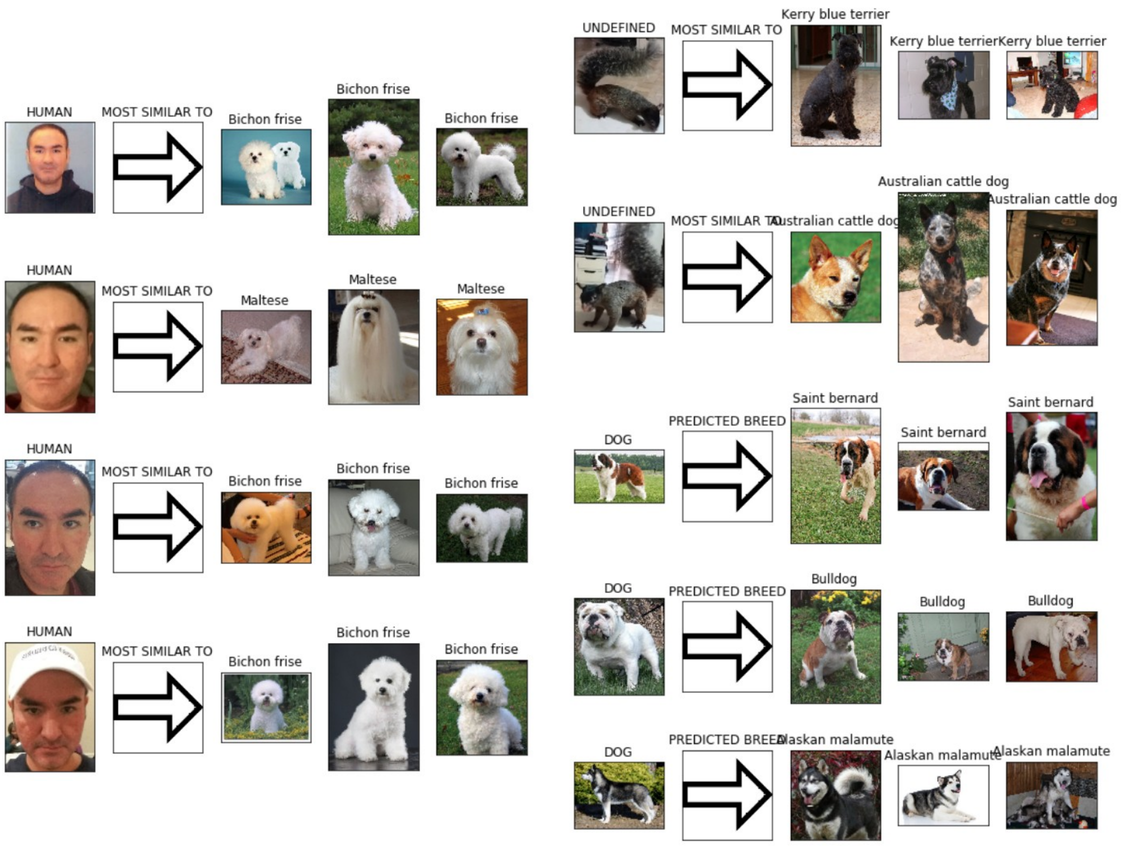
The best ResNet-50 model was saved and produced the following results in the test dataset:

test_loss=0.009719, test_acc=85.05% (711/836)

The final solution (test_acc=85.05%) is much better than the Benchmark model (test_acc=15.07%).

Part 7. Creating and testing the Dog App.

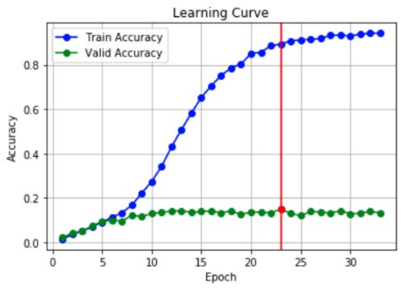
The human face detector, the dog detector, and the ResNet-50 are combined to create the Dog App. Each photo is classified into 3 categories: Human, dog, and other. Then, each photo is classified into one of 133 dog breeds. The results are amazing!



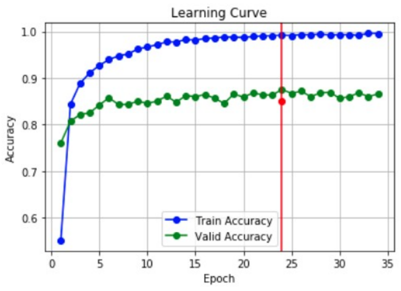
Refinement

I refined the model by using a validation dataset in order to constantly evaluate the validation accuracy of the training model at each epoch. The training model with maximum accuracy in the validation dataset was selected and saved for future usage.

Benchmark CNN (from scratch)



ResNet-50 (transfer learning)



Regarding the ResNet-50 (transfer learning), I didn't have problems because I have a lot of experience with convnets and transfer learning. (I have a M.Sc. in Machine Learning from Georgia Tech and I work as a project reviewer for the Computer Vision nanodegree at Udacity.) I decided to use the best hyperparameters by intuition, the hyperparameters that worked well in my previous projects. So, I didn't make too many experiments to obtain very good results. The Adam optimizer with default parameters helped a lot to obtain great results without too much experimentation.

However, it was a little bit hard to consistently obtain a test accuracy greater than 10% in the Benchmark CNN. I played with many hyperparameters for the convolutional layers and the fully-connected layers. My test accuracy was between 8% and 11%. And it varied randomly, depending on the random numbers of specific runs. Then, I found the sweet spot after many experiments and I got a **test_acc=15.07%**

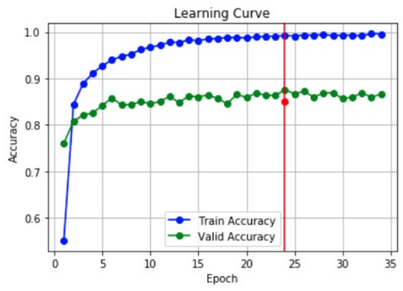
An interesting experiment is to vary the hyperparameter MAX_END_COUNTS. It controls the amount of epochs without improvement in the validation accuracy required to end training. With MAX_END_COUNTS=5, the model underfit, resulting in bad predictions that didn't resemble the photos. With MAX_END_COUNTS=20, the model overfit, resulting in very good predictions that really resemble the photos. However, due to the overfitting, my 4 photos produced 4 different dog breeds. The sweet spot was MAX_END_COUNTS=10 because my 4 photos produced only 2 dog breeds that are very similar. The same applies to the 2 photos of my black squirrel. The testing accuracies for dog photos drop with underfitting and overfitting. That's why I selected MAX_END_COUNTS=10.

IV. RESULTS

Model Evaluation and Validation

I selected the final model by using a validation dataset in order to constantly compute the validation accuracy of the training model at each epoch. The training model with maximum accuracy in the validation dataset was selected and saved for future usage. When the amount of epochs without improvement in the validation accuracy is greater than MAX_END_COUNTS=10, training ends.

ResNet-50 (transfer learning)



The final solution, the ResNet-50 with pretrained weights and transfer learning, produced its best results at the epoch 24:

train_loss=0.000749, train_acc=99.19%
valid_loss=0.006908, valid_acc=87.54%

The best ResNet-50 model was saved and produced the following results in the test dataset:

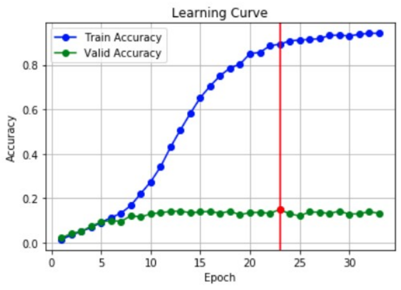
test_loss=0.009719, test_acc=85.05% (711/836)

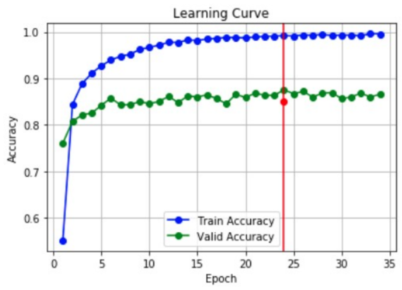
This model generalizes well to unseen data because the valid_acc = 87.54% and the **test_acc = 85.05%**, which is a very good result. The test dataset in this project is totally unseen and never intervened in the training process or in the metaoptimization of hyperparameters. Therefore, the results obtained by the model can be trusted.

When I took the PyTorch Scholarship Challenge from Facebook and Udacity, I programmed as the final project a VGG19 with pre-trained weights and transfer learning. I obtained 96% accuracy in the test dataset. And I extracted the test dataset by removing some elements from the dataset of images, in order to keep the test dataset untouched and unseen. In spite of the fact that I removed some elements from the dataset of images, meaning that the training dataset and validation dataset had significantly less elements, the results were almost the same. This suggest that small perturbations in training data or the input space don’t affect the results greatly. Moreover, extending the training dataset by using sensible transformations makes the pattern recognizer more robust and more invariant to changes. The same applies to the ResNet-50 because it’s a convnet like VGG19.

Justification

In this section, we will compare the Benchmark CNN’s performance against the ResNet-50’s performance.


Benchmark CNN (from scratch)


ResNet-50 (transfer learning)

The benchmark model produced its best results at the epoch 23:
train_loss=0.005845, train_acc=89.31%
valid_loss=0.104890, valid_acc=14.97%

The best benchmark model was saved and produced the following results in the test dataset:
test_loss=0.120632, test_acc=15.07% (126/836)

The final solution, the ResNet-50 with pretrained weights and transfer learning, produced its best results at the epoch 24:
train_loss=0.000749, train_acc=99.19%
valid_loss=0.006908, valid_acc=87.54%

The best ResNet-50 model was saved and produced the following results in the test dataset:
test_loss=0.009719, test_acc=85.05% (711/836)

It is a really humbling experience to try to do your best at creating the best convnet possible with the best hyperparameters and yet the results pale in comparison to the results of the ResNet-50 with pretrained weights and transfer learning.

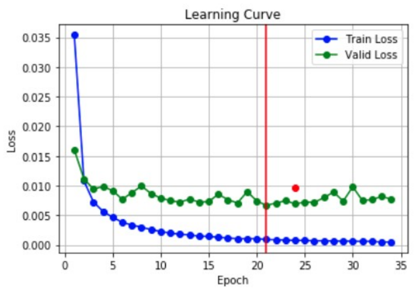
We clearly see that the ResNet-50’s performance is much stronger than the Benchmark CNN’s performance. The final solution with a test accuracy of 85.05% is not significant enough to have fully solved the problem in a complete way. It is just good enough. To fully solve the problem, a test accuracy of 95% or more is needed. I have achieved such level of test accuracy (96%) by using the VGG19 with pre-trained weights and transfer learning, when I took the PyTorch Scholarship Challenge from Facebook and Udacity.

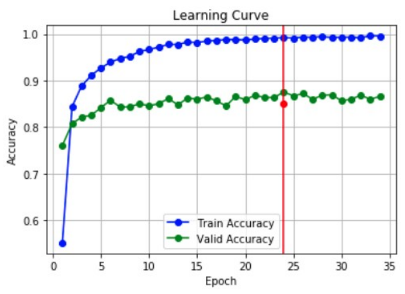
V. CONCLUSION

Free-Form Visualization

Two important visualizations to emphasize in this project are the learning curves and the results of the Dog App.

There are 2 types of learning curves: Loss versus Epoch. And Accuracy versus Epoch. We can optimize for either the minimum validation loss or the maximum validation accuracy. In this project, we optimize for the maximum validation accuracy, the second graph on the right, below.





The final solution, the ResNet-50 with pretrained weights and transfer learning, produced its best results at the epoch 24:

train_loss=0.000749, train_acc=99.19%
valid_loss=0.006908, valid_acc=87.54%

The best ResNet-50 model was saved and produced the following results in the test dataset:

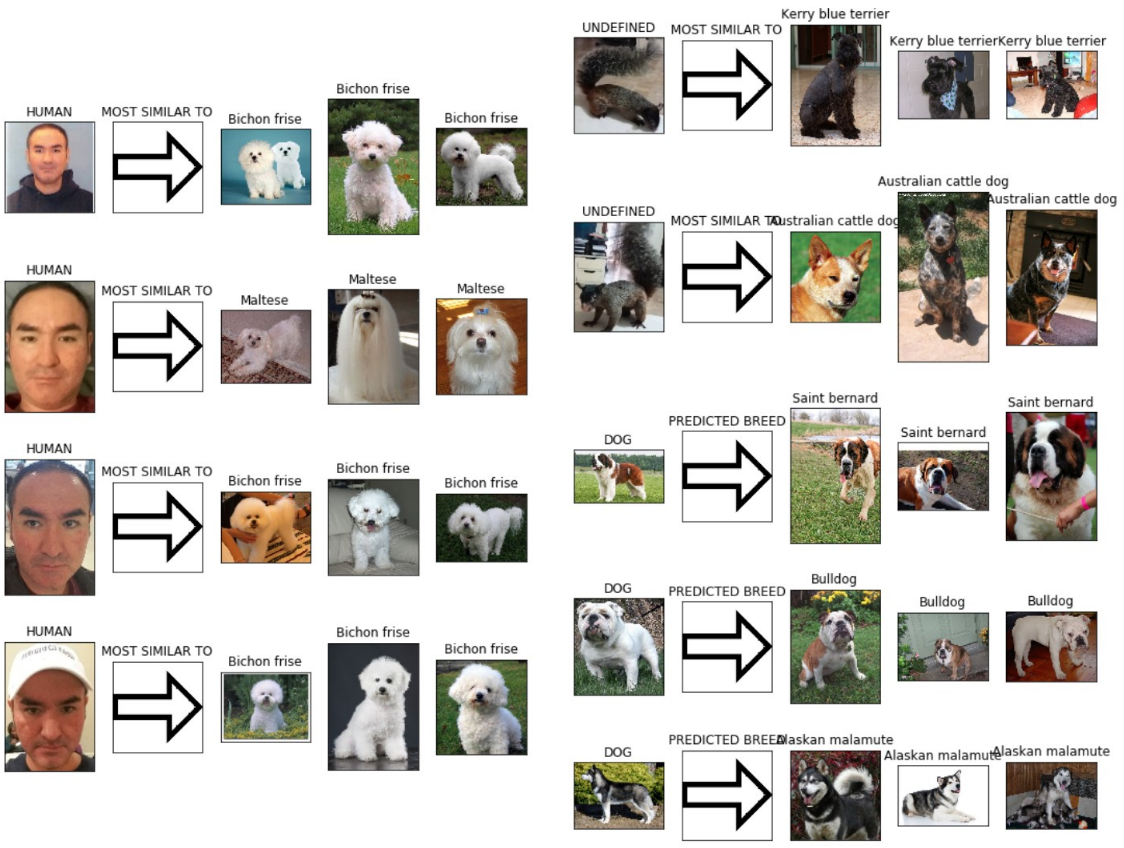
test_loss=0.009719, test_acc=85.05% (711/836)

These 2 points **test_loss** and **test_acc** at epoch 24 were drawn in red in the corresponding graphs above.

The results of the Dog App are also interesting. 4 photos of me were shown to the Dog App. And my 4 photos were classified as HUMAN and as white cute dogs: Bichon Frise and Maltese. I think they probably reflect some visual aspects of my personality.

2 photos of my black squirrel Negrita were shown to the Dog App. And the 2 Negrita's photos were classified as UNDEFINED and as 2 similar dog breeds: Kerry Blue Terrier (black) and Australian Cattle Dog (gray). My squirrel is somewhat black and somewhat gray.

3 unseen photos of real dogs were collected from outside sources and were shown to the Dog App. I searched them on Google with the words: Saint Bernard, Bulldog, and Husky. The 3 dog photos were classified as DOGS and as the corresponding dog breeds: Saint Bernard, Bulldog, and Alaskan Malamute. They were accurate predictions, even the Alaskan Malamute because the dog breed Husky does not exist in the dataset. So, the convnet managed to route it to the most similar dog breed, which is really amazing.



Reflection

In summary, the Dog Breed Classification project has the following steps:

1. Importing the Human Dataset.
2. Importing the Dog Dataset.
3. Testing the human face detector.
4. Testing the dog detector.
5. Benchmark CNN (from scratch).
6. ResNet-50 (transfer learning).
7. Creating and testing the Dog App.

The most interesting aspects of the project are:

- how common patterns of the visual field can be transferred and reused by the convnet;
- and how convnets can extrapolate patterns outside their intended domain and can make visual analogies.

The most difficult aspect of the project was to improve the test accuracy of the Benchmark CNN and how humbling it is to compare these poor results against near state-of-the-art results obtained by the ResNet-50 and the VGG19.

I think I'm satisfied with the final results. It's so satisfying the see the results of near state-of-the-art convnets. My personal motivation to select this project is that I love animals. I could observe the behavior of animals endlessly. While, some scientists prefer to look at the stars, planets, black holes, and the universe, I'm rather passionate about the inner universe of our minds and the minds of animals. I've been studying neuroscience and artificial intelligence for many years. And such experience gave me the tools to understand animal behavior in a very deep way. I always learn new things from my observations and experiences. For example, those beautiful photos of dogs show a lot about the personalities of dogs. I wonder if the convolutional neural networks of this project can extract such personality traits. I guess the answer is yes. However, analyzing such artificial insights goes beyond this simple classification project. But this project is a great opportunity to think about those deep questions.

Improvement

The final solution with a test accuracy of 85.05% is not significant enough to have fully solved the problem in a complete way. It is just good enough. To fully solve the problem, a test accuracy of 95% or more is needed. I have achieved such level of test accuracy (96%) by using the VGG19 with pre-trained weights and transfer learning, when I took the PyTorch Scholarship Challenge from Facebook and Udacity.

Convnets produce state-of-the-art results. However, I think I could create the generalization of convnets based on Abstract SLAM, my new cognitive network. I will publish some papers about it soon. If I used my final solution as the new benchmark, I could find a better solution by using the neural networks I mentioned above.

Bibliography

LeCun, Y. and Bengio, Y. (1995). Convolutional networks for images, speech, and time series. In Michael A. Arbib (ed.), Handbook of Brain Theory and Neural Networks. MIT Press. pp. 3361

Goodfellow, I., Bengio, Y., and Courville, A. (2016). Deep Learning (Adaptive Computation and Machine Learning series). MIT Press.

Mitchell, T. (1997). Machine Learning (McGraw-Hill International Editions Computer Science Series). McGraw-Hill.

Russell, S. and Norvig, P. (2020). Artificial Intelligence: A Modern Approach (4th Edition). Pearson.

Viola, P. and Jones, M. (2001). Rapid Object Detection using a Boosted Cascade of Simple Features.