ADENIYI ADEBOYE (MACHINE LEARNING NANODEGREE)

DOG BREED CLASSIFICATION PROJECT'S REPORT

I. Definition

Project Overview

Dog breed classification falls under fine grained image classification which entails distinguishing fine-level image categories in images i.e. classifying images with high interclass similarity and large intraclass variations such as bird species, dog breeds and airplane types to mention but a few. In other words, there are comparatively few differences between breeds and relatively large differences within breeds, differing in size, shape, and color.

Traditional machine learning approaches were utilized in the nineties to classify images, but this yielded very low accuracies and were faced with several challenges such as hand-crafted feature extraction process. In recent years, deep neural networks such as Convoluted Neural Networks (CNN or ConvNET) have been heavily relied on for image classifications, object detection, image captioning and localization simply because of their ability to find complex formation and underlying patterns / features in large training dataset such as images.

For this project, CNN will be utilized for the dog breed classification. As described in previous research papers, CNN is a type of multi-layer neural network inspired by the mechanism of the optical system of living creatures. CNN architectures make assumption that the inputs are images (they take into account the spatial dimension of the input), which allows us to encode certain properties into the architecture. A typical CNN is composed of a single or multiple blocks of a convoluted layers and sub-sampling or pooling layers, one or more fully connected layers and finally an output layer (figure 1).

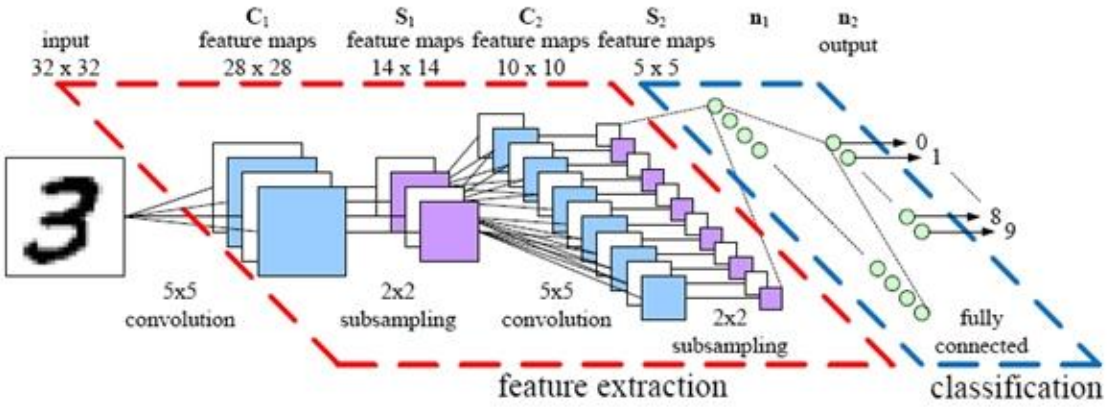


Figure 1: A convoluted neural network built for a handwritten digit classification. From the left: we have the handwritten image(3), convoluted layer (with four feature maps), pooling layer(four feature maps), another convoluted layer (with 12 feature maps), another pooling layer (with 12 feature maps), these convoluted and pooling layers are basically used for feature extraction. followed by them, fully connected layer i.e. each neuron in this layer is connected to all activation values from the previous layer, then the output layer that has the

number of classes that we want to predict. Note that the filter size for the convoluted layers is 5 x 5, max pooling filter is 2 x 2.

Motivation and Problem Statement

Diverse breed of dogs exist in today's world and due to their ubiquity, they have been found to be one of the most difficult breeds to identify. This motivates the idea of building a deep learning model that is capable of identifying dog breeds with high accuracy. Furthermore, this model can also help to identify breeds of cats, horses, species of birds and plants, or even models of cars.

The aim of this project is to a build a fine-grained image classification model which classifies dog breeds given a specific number of different dog classes. To accomplish this, the following objectives were carried out:

1. ***Importation of the Dog dataset and Data Exploration*** to visualize the statistical distribution of the dog samples within each dog breed and among the training data – the data used to train the model, validation data – data used to evaluate the model while training, and test data – data used to evaluate the overall performance of the model.

2. ***Detection of Human faces and Assessment of Human face detector*** – Here, we implemented a human face detector algorithm and tested it on human files and dog files to evaluate its performance.

3. ***Implementation of a pre-trained model on dog dataset*** – A pre-trained model was obtained and further was used to make predictions on the dog dataset, this was implemented and was utilized in detecting the percentages of dog images in dog files and human files.

2. ***Data loading*** – the dog dataset (images) are specifically loaded batch by batch into the neural network, this has to be sorted in sequential manner before proceeding with the data pre-processing.

3. ***Data Pre-processing*** – this means preparing the data to have the appropriate sizes, format (tensor) and dimensions that is required for an input data that is to be passed into a neural network.

4. ***Building the CNN from scratch*** – this requires selecting and tweaking the right parameters to be used to build the CNN model, selecting the right amount of convolutional layers, max-pooling layers, batch normalization layers, activated layers, dropout layers, fully connected layers and the final output layer, you intend to have within your network. The building of a CNN is problem specific you can decide to build your neural network based on your understanding of the problem and the knowledge of how CNN works for several image classification challenges.

5. ***Train your Model*** – this involves selecting the amount of epochs i.e. the amount of times you want your whole training data to be trained across the network. Training your model also

entails closely monitoring the training and validation losses after each epoch as your model is trained across the CNN iteratively.

6. ***Evaluate your model*** – After training your model on your training data and monitoring its loss closely on both the validation and training loss for the purpose of avoiding overfitting, you will evaluate it on the test (Unseen) data to check if your model actually learnt something while training and can generalize well on out-of-sample data.

If the model built from scratch has an accuracy of > 10%, this means our dog classification problem is highly solvable and can be optimized using transfer learning.

7. ***Build the CNN through the use of Transfer Learning*** – this entails selecting and using a pretrained model to train your model through applying and transferring the pretrained model's knowledge to your own train data, this also entails selecting right parameters for the model's implementation. The selection of a pretrained model is based on how related the model is to your image classification problem, there are lots of state-of-the-art models that can be utilized for this purpose, selecting the right one is based on our own preference.

Furthermore, for the pretrained model most of its layers were left frozen (not trained) since only the last part / layers of the network required fine-tuning / training.

8. ***Train your model (that was built on a pre-trained network)*** – this entails selecting the amount of time (epochs) you want your model to be train for, and further monitoring the training and validation losses, in our case the model with the lowest validation loss was saved.

9. ***Evaluate your model*** – this entails testing your trained model on unseen data, to be able to know how well the model was able to generalize what it learnt from the training data to an unseen data.

In our case, if the accuracy is less than 60% this means the model needs to be retrained as it hasn't been fully optimized for our classification problem.

10. ***Build an algorithm for Dog breed and human face identification*** – An algorithm is built that takes in an image path and classifies whether an image contains a human face or a dog, if it contains a dog, it further predicts which dog breeds the dog belongs.

The intended solution of the itemized objectives stated above is for the model built with the help of transfer learning to be able to classify and associate a dog image to one of the dog breed classes predicted by the model.

Evaluation metrics

The measure of our model performance will be based on the model's accuracy score (in percentage). The true measure of the model's performance is based on the test accuracy, which represent the trained model classifying on completely (never seen before) new images.

While training, the loss is also a vital parameter to watch out for, if it is decreasing incrementally during training then the model is actually learning from the training data but if otherwise then you should consider monitoring the model closely, and if further persist, you should stop the model. In our case, the training and validation loss were computed

simultaneously as the model training was on-going, and the model with the lowest validation loss was saved.

II. Analysis

Datasets

The dataset to be utilized for this project has already been downloaded in the Udacity 'data' folder and was imported into the Dog breed's project workspace for analysis.

Data Exploration

The dog dataset contains 8351 images with 133 different classes. This dataset was split into three data directories namely: train, validation and test data. The table below shows summary of the statistics of the dog breed dataset.

	Train Data	Test Data	Val. Data
count	133.000000	133.000000	133.000000
mean	50.225564	6.285714	6.278195
std	11.863885	1.712571	1.350384
min	26.000000	3.000000	4.000000
25%	42.000000	5.000000	6.000000
50%	50.000000	6.000000	6.000000
75%	61.000000	8.000000	7.000000
max	77.000000	10.000000	9.000000

The above descriptive statistical table tells us the following regarding our data:

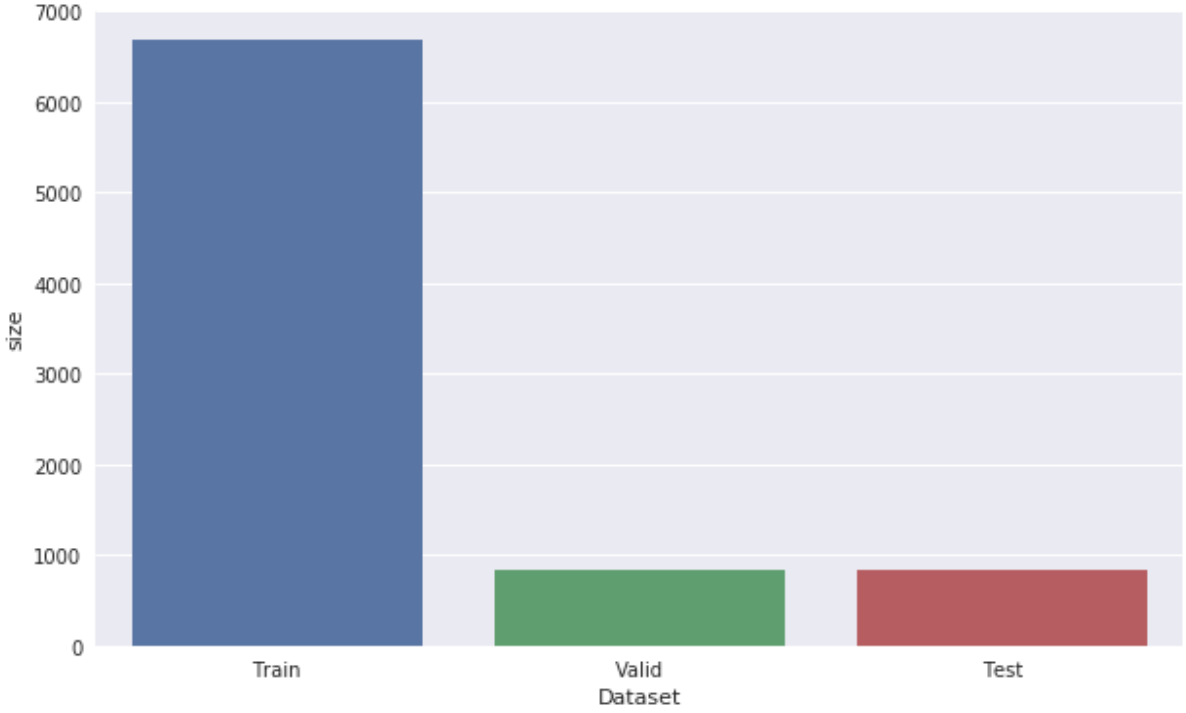
- The number of dog breed classes in each dataset == 133.
- The average number of samples in a specific class for the train data == 50, validation data == 6 and for the test data == 6.
- The minimum number of sample in a specific class in the training data == 26, validation data == 4, and for test data == 3.
- The maximum number of samples in a specific class in the training data == 77, validation data == 9, and for test data == 10.

In addition, the human files contains 13233 images that would be utilized in the human face detection.

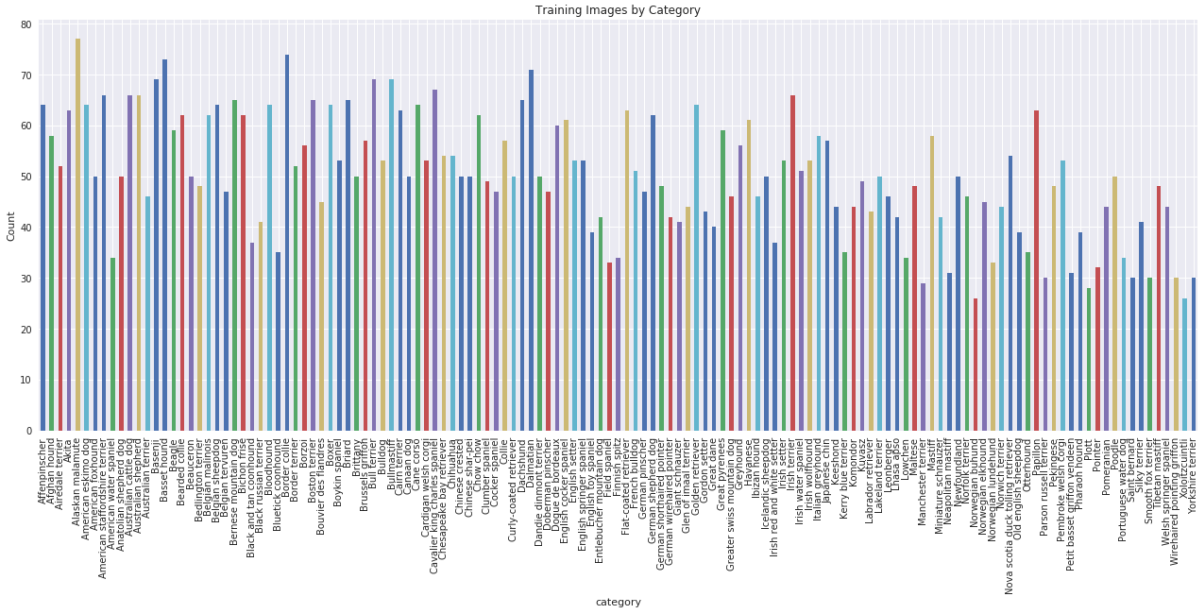
Exploration Visualization

In this section:

- we will visualize the distribution of the dataset among the train, validation and test data, which is shown below:



- Here, we will also take a look at the distribution of the different dog classes across the train data:



- Finally, we will take a look at the first batch of the dog images (20 images per batch) that is to be passed into the neural network, ***each dog image with its class names in this grid is written below***:

From the top left (dog images with their class names)*: 1 – English Setter, 2 – English Toy Spaniel, 3 – Australian Shepherd, 4 – Basenji, 5 – Airedale Terrier, 6 – Flat-coated retriever, 7 – Italian greyhound, 8 – Belgian malinois, 9 – Airedale Terrier, 10 – Beagle, 11 – German Pinscher, 12 – Ibizan hound, 13 – Ibizan hound, 14 – Alaskan Malamute, 15 – Cane corso, 16 – Bulldog, 17 – Belgian Tervuren, 18 – Miniature Schnauzer, 19 – Japanese Chin, 20 – Cavalier King Charles Spaniel* ***(Bottom right).***

Algorithm and Techniques

We will be using a deep learning framework called ***Pytorch*** for building the convoluted neural network that will be utilized for this project. ***Pytorch*** gives you uttermost control over your CNN architecture allowing you to build complicated models sequentially with relative ease. The building of the CNN architecture was done in Python.

The optimization algorithm used for updating the parameters was Adam optimizer, regularization method used to reduce overfitting was Dropout and the activation function utilized was Relu max(0,x) (Rectified linear unit), all of these will be re-instated and discussed in the methodology section.

III. Methodology

In this section, we will start with discussing the data loading, followed by data pre-processing, then, the steps involved in building our model from scratch (which serve as our benchmark model) and the model built through transfer learning and finally implementing the model.

Data Loading

This entails first getting the train, validation and test data into image folders, then extracting 20 images per batch (batch size) from each of these folders into the neural network for every iteration until the data in these folders are exhausted. This is required when working with neural networks in pytorch as the network expects input tensors to be in batches.

The batch size can vary depending on your discretion but for this project 20 images per batch is used.

Data Pre-processing

Data pre-processing was done with the help of the ***transforms operations in pytorch*** before the commencement of building the CNN model, and these pre-processing methods are highlighted below:

- **transforms.Resize(256)** - this resizes the smallest edge of the images to a 256 size which is much needed before we centre crop.
- **transforms.Centrecrop(224 x 224)** - this scales down all the images to 224 x 224 (height x width) which is required when preparing our images for the CNN. It is also needed before we perform transfer learning because the deep CNN that were used for the ImageNet data requires all image tensors to have a 224 x 224 size.
- Data augmentation was done to artificially increase the number of training images our model sees through the use of random transformations:
- **transforms.RandomHorizontalFlip(p=0.5)** - horizontally flip the given PIL image randomly with a given probability.
- **transforms.RandomRotation(10)** - randomly rotates the PIL images between +10 and -10 degrees.
- The main idea behind Data augmentation is to help reduce overfitting this is because as our network is exposed different variations/distortions of the input image tensors it helps improve its performance in understanding the underlying patterns of the images which further helps in better generalization to out-of-sample test data.
- *It is IMPORTANT to note that data augmentation is only done during training (on train data), this is because you do not want to add noise or distort your test / real world sample image(s).*
- Normalization was performed on the input image **transforms.Normalize(mean = [0.485, 0.456, 0.406], std = [0.229, 0.224, 0.225]** this serves to make convergence happen quicker during training based on various research in CNN.
- **transforms.ToTensor()** - this transforms the input image into an image tensor with is required when working with neural networks.

Steps involved in building the benchmark model (Our model built from scratch)

1. Five (5) convolutional layers were chosen for our CNN architecture based on previous literatures on dog breed classification, the structure includes:

- Conv_layer1: input channels(3), output channels(16)
- Conv_layer2: input channels(16), output channels(32)
- Conv_layer3: input channels(32), output channels(64)
- Conv_layer4: input channels(64), output channels(128)
- Conv_layer5: input channels(128), output channels(256)

***Note that**: Input image tensors: 3, 224 x 224 (224 x 224 pixels in the (3) RGB colour channels); filter size for the whole CNN architecture (3 x 3), stride = 1, Padding = 0 (i.e. No zero padding was done).*

2. After every convolutional layer, Batch Normalization was done to normalize the input layers to have a mean == 0 and variance == 1. This is done with respect to the research paper on Batch Normalization: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In this paper, it was highly recommended to add the BN transform immediately before the non-linearity and furthermore, BN transform applies the same linear transformation to each element in a given feature map.

3. After the batch normalization, Activation function called Relu (- Rectified linear unit max(0,x)) was used to activate the normalized neurons, which basically converts incoming inputs less than 0 to 0 and those higher to their exact number.

4. After performing non-linearity on the normalized neuron, Max pooling layer (2,2) was used to downsample the number of neurons in each feature maps(i.e. taking the maximum weighted value within a 2 by 2 block). This means if the incoming (activated and normalized) convolutional layer has 16 feature maps/channels/depths, max pooling layer is applied to each and every one of them.

5. After the fifth (and final) Max pooling, Dropout(0.2) is applied to the pooling layer, to further reduce overfitting by randomly (and temporarily) deleting about 20% of the neurons in the layer.

6. Finally, we pass the input neurons (in which 20% of it has been dropped out) to a Fully connected (FC) layer(256x6x6) and then this FC layer outputs the 133 intended classes of the dog breeds.

Steps involved in building the of the model with transfer learning

First of all, I used ResNet 50 as my pretrained model, This model has 50 parameter layers that was optimized and utilized for the ImageNet classification challenge achieving 6.71% top-5 error rate in 2015 Resnet research paper -- He et al., 2015 (figure 2).

The logic behind the ResNet CNN structure is that it performs residual learning to each of the stacked layers in its DNN architecture by using shortcut connections to perform identity mapping to these individual layers (- where the identity (x) that is to be learnt by a layer is recast into it after the layer has been activated i.e. F(x) + x, this basically is residual mapping where each layer is directly fitted to a desired underlying pattern / identity).

- The architecture of the ResNet 50 consist of Convolutional layer, Batch normalization layer, Relu activation function layer, Average pooling layer and finally a Fully connected layer all of which are arranged based on the discretion of the network creators and developers.
- For the purpose of transfer learning, All the layers *prior t*o the last fully connected (FC) layer in the network *were FROZEN i.e. we didn't train them***.**
- We replayed the last fully connected layer output with 133 output neurons leaving its input neurons as 2048.
- This means that we are going to train only about 272,000 weights and 133 biases of the last FC layer for this transfer learning model.

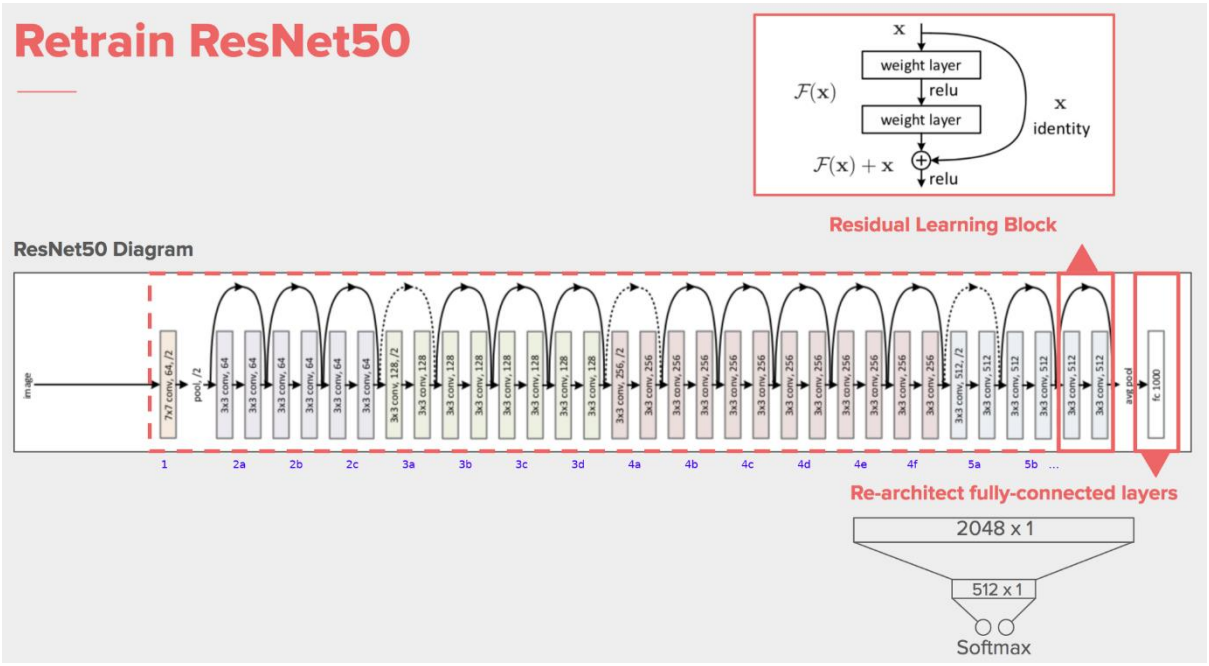


Figure 2: CNN architecture of the ResNet 50, He et al., 2015. On the top right corner is the descriptive illustration of the residual learning that involves mapping an identity (x) to an activated layer F(x) resulting to F(x) + x.

Implementation of the model built from scratch

First of all, the parameters utilized for the implementation of this model includes Adam optimizer – used to update parameters after every backpropagation (computation of the cost function gradient with respect to weights and biases), Cross Entropy loss – was used to compute the cost function for this specific model, and finally learning rate of 0.001 was used to take steps for the gradient descent in order for our model to reach the global minimum and converge effectively.

Our model was implemented using the aforementioned parameters and the training and validation loss was monitored closely while the model was training, in addition, the model with the lowest validation loss was saved while training was going on. The model was trained for 10 epochs as it was observed that this was sufficient for the model to learn from the training data, and increased number of epochs would result to overfitting to the training data.

The tabular description below shows how the loss was varying during model training ***(check the python notebook to verify this):***

```
Epoch: 1        Training Loss: 0.000425        Validation Loss: 0.004502
The best validation loss is presently: 0.004502. Saving model...
Epoch: 2        Training Loss: 0.000414        Validation Loss: 0.004491
The best validation loss is presently: 0.004491. Saving model...
Epoch: 3        Training Loss: 0.000408        Validation Loss: 0.004501
Epoch: 4        Training Loss: 0.000403        Validation Loss: 0.004395
The best validation loss is presently: 0.004395. Saving model...
Epoch: 5        Training Loss: 0.000400        Validation Loss: 0.004481
Epoch: 6        Training Loss: 0.000396        Validation Loss: 0.004419
Epoch: 7        Training Loss: 0.000392        Validation Loss: 0.004533
Epoch: 8        Training Loss: 0.000382        Validation Loss: 0.004428
Epoch: 9        Training Loss: 0.000382        Validation Loss: 0.004412
Epoch: 10       Training Loss: 0.000374        Validation Loss: 0.004451
```

Note that after epoch 4 the validation loss was varying, i.e. not decreasing as seen in the training loss, this could be an indication of overfitting. We trained for 10 epochs because we further experimented with 20 – 30 epochs and realized that the loss was not decreasing in a declining manner as a result we fell back on only training for 10 epochs. When we evaluated the model using the test data it gave a sufficiently good accuracy to start with (at least above what we were expecting), before optimizing the model using pre-trained model.

Implementation of the model with transfer learning

First of all, the parameters utilized in the model built from scratch was also used for the implementation of this model and they include the Adam optimizer – used to update parameters after every backpropagation -- (which is computation of the cost function gradient with respect to weights and biases), Cross Entropy loss – was used to compute the cost function for this specific model, and finally learning rate of 0.001 was used to take steps for the gradient descent in order for our model to reach the global minimum and converge effectively.

For the model built with transfer learning only the fine-tuned layers were trained i.e. the last fully connected layer with 2048 input neurons and 133 output neurons representing the dog classes to be predicted. The model was also trained for 10 epochs and the training and validation loss were closely monitored during model training. In addition, the model with the lowest validation loss was saved during model training.

The tabular description of how the loss was behaving during the model training is shown below ***(check the python notebook for verification***):

```
Epoch: 1        Training Loss: 0.000122        Validation Loss: 0.001329
```

```
The best validation loss is presently: 0.001329. Saving model...
Epoch: 2        Training Loss: 0.000083        Validation Loss: 0.001126
The best validation loss is presently: 0.001126. Saving model...
Epoch: 3        Training Loss: 0.000071        Validation Loss: 0.001207
Epoch: 4        Training Loss: 0.000058        Validation Loss: 0.001074
The best validation loss is presently: 0.001074. Saving model...
Epoch: 5        Training Loss: 0.000050        Validation Loss: 0.001142
Epoch: 6        Training Loss: 0.000051        Validation Loss: 0.000882
The best validation loss is presently: 0.000882. Saving model...
Epoch: 7        Training Loss: 0.000045        Validation Loss: 0.001030
Epoch: 8        Training Loss: 0.000042        Validation Loss: 0.001017
Epoch: 9        Training Loss: 0.000041        Validation Loss: 0.001069
Epoch: 10       Training Loss: 0.000036        Validation Loss: 0.001006
```

As observed above, after epoch 6 the loss was not decreasing in a declining manner rather it was fluctuating, we further experimented by training the model for 30 epochs but it was not decreasing in a manner we would expect, as a result we fell back to just training for 10 epochs which ended up giving us a high accuracy when we evaluated the model using the test data.

Refinement

Several parameters were tweaked and optimized for the model to yield high accuracy and lowest possible error. For instance, at the initial stage of the model training, the SGD (Stochastic Gradient Descent) optimizer was used but this didn't work well with the model as the loss was not declining instead it was just fluctuating for over 20 epochs. As a result, we used Adam optimizer which gave a better result.

Furthermore, we started with 0.01 – learning rate, but we observe the error rate were not also behaving well, we were observing abnormal error values i.e. error values were skyrocketing after each epoch, as a result of this we decided to fall back using 0.001 – learning rate, which instantaneously worked well and resulted in the error rate behaving normally.

IV. Result

Model Evaluation

First of all, the model built from scratch was evaluated on the test data based on its accuracy, as it is stated in the notebook (by Udacity) that if the accuracy is less than 10% that means the model is not performing well i.e. it has not learnt the underlying patterns in the training data and as a result won't generalize well to unseen data.

However, our model built from scratch when evaluated on the test data had 21% accuracy and test loss of 3.55, which indicated that the dog classification problem is to some extent solvable by our benchmark model. This result also showed us that with pre-trained model we would expect a high accuracy relative to our built from scratch model.

Comparatively, we evaluated our model built using transfer learning on the test data, and we had 87% accuracy and 0.86 test loss, this showed a high increase in accuracy relative to the model built from scratch and a significant drop in the loss.

Overall, this showed that ResNet 50 is well suited for dog classification problem because it was successfully able to transfer its knowledge from the ImageNet classification challenge to our small dog breed classification problem.

Justification

To further re-state the point made earlier that our optimized model built using transfer learning can be able to generalize well to unseen data, we built an algorithm that takes in an image path, check if the input image is a human face or dog picture, if it is a dog picture, it predicts the dog's breed.

This algorithm worked perfectly well for distinguishing between dog images of similar classes giving to it, for example:

The model predicted this dog image as Mastiff, which is said to be right.

The model predicted this dog image as Bullmastiff, which is said to be right.

V. Conclusion

Reflection

The major highlight of this project is that convolutional neural network is an amazing deep learning tool for computer vision especially image classification, as it is able to train models to generalize well to out-of-sample images, classifying them correctly with minimal errors if trained well.

In addition, one of the key lessons from this project is that training your model with a pre-trained state-of-the-art model will make it easy for you to use your model for small dataset image classification problems, because these pre-trained model with their deep layers are optimized to get you the lowest possible error rates and highest possible accuracy as seen in this project.

The only limitation to this project, is the small dataset of the dog breeds, as it is known that deep neural networks are extremely data hungry so feeding them with small data samples will result to them overfitting on your training data and not generalizing well on your out-of-sample data.

Overall, ResNet 50 which was used as our pre-trained model is well suited for dog classification problem because it was successfully able to transfer its knowledge from the ImageNet classification challenge to our small dog breed classification problem and I further believe it can be applied to other fine-grained classification problem.

Improvement

The following points of improvement would be considered in future when giving similar task:

- More training dataset will be required to improve the algorithm and reduce overfitting, this is because DNN are data hungry -- Increased training data improves their generalization to unseen data.
- State-of-the-art models with deeper layers could be further utilized to improve our model performance, as it is shown in various research papers that deeper networks yield top error rates in image classification challenges.
- In addition to our model built from scratch (which serve as our baseline model), other benchmark models could be developed to compare and contrast how different models (based on their architectures and parameters) generalize to out-of-sample images.

References

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385, 2015b.

Xavier S. Higa, 2019. Dog Breed Classification Using Convolutional Neural Networks: Interpreted Through a Lockean Perspective. **Lake Forest College Publications.** Senior Theses Student Publications, 4-7-2019. https://publications.lakeforest.edu/seniortheses.

Pratik Devikar, 2016. Transfer Learning for Image Classification of various dog breeds. *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET) Volume 5, Issue 12, December 2016. ISSN: 2278 – 1323*

Sachin Padmanabhan, 2016. Convolutional Neural Networks for Image Classification and Captioning.

Sergey Ioffe and Christian Szegedy, 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. 1502.03167v3 [cs.LG] 2 Mar 2015

Whitney LaRow, Brian Mittl, Vijay Singh, 2016. Dog Breed Identification.