|  |  |
| --- | --- |
| Dog BreedÂ Classifier  dog Breed Classifier | BY  Sai Aditya varma Hemanth  Yi Yao Yan Wang |
|  |  |

**Introduction**

In this project, we will identify breed of dog from the pictures of dog using convolutional neural network.

A convolutional neural network is a class of deep neural networks, most commonly applied to analyzing visual imagery.

CNNs are regularized versions of multilayer perceptron’s. Multilayer perceptron’s usually refer to fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. The "fully-connectedness" of these networks make them prone to overfitting data. Typical ways of regularization include adding some form of magnitude measurement of weights to the loss function. However, CNNs take a different approach towards regularization: they take advantage of the hierarchical pattern in data and assemble more complex patterns using smaller and simpler patterns. Therefore, on the scale of connectedness and complexity, CNNs are on the lower extreme. Unlike fully connected networks which uses flatten layers which destroy spatial knowledge of data, CNN’s maintain this information for the development of higher-level filters which are good at classification.

A convolutional neural network consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of convolutional layers, RELU layer i.e. activation function, pooling layers, fully connected layers and normalization layers.

**Dataset**

The Stanford Dogs dataset contains images of 120 breeds of dogs from around the world. This dataset has been built using images and annotation from ImageNet for the task of fine-grained image categorization.

The dataset we are using is provided with a set of images of dogs. The dataset comprises 120 breeds of dogs images in sub directories all labeled, which is 120 classes. Also, we totally have 20580 images that we are going to take in our procedure.

Besides, the number of training images per class is varied from 40-100. This dataset comes with in Images folders. we transferred the dataset into train, test, validation folders.

Train folder is for training convolutional net,

Validation is to finetune hyper parameters,

Test is for testing our model how well it performs on new data set.

Link: Stanford Dogs Dataset (http://vision.stanford.edu/aditya86/ImageNetDogs/).

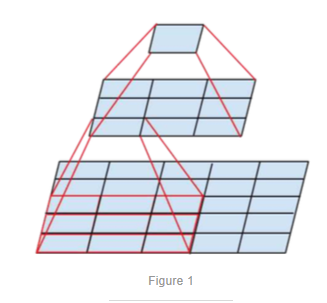
**Network Architecture**

We aim to use CNN architecture.

Convolutional layers are partially connected, and extracts feature from images using kernel which in our case 2\*2 size, the filter pass through image horizontally and vertically convoluting the pixels and weights obtained by back propagation.

Initially we started with 16 filters and experimented going up to 256 filters.

In order to avoid overfitting and to have a ease of computation we choose max pooling layers with kernel size 2\*2 and with strides =2 which decreases the spatial dimension by 2 if padding is set to same.



By doing this we are extracting high level feature from images by increasing depth and decreasing spatial dimensions so at the last features we obtain at max pooling are the filters which can recognize breed information in our data set.

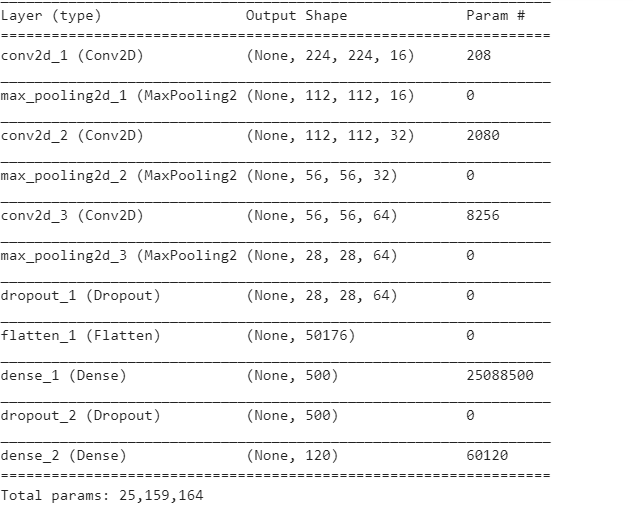
Once we obtain features we need to classify it so we applied flatten layers such that all the features are obtained in one array so that we can pass it through deep neural network for classification.

Before doing that there is high chance of overfitting since the inputs to the deep network are 50000 nodes in our case, so we must choose perfect regularization method. we choose to add dropout layer with input 0.3 such that 30% of nodes will be disconnected randomly while training deep network.

Experimented with 3 layers of dense layers , starting from 500 hidden nodes went to 120 nodes for output layer. used SoftMax as activation in the last layer for classification.

Number of filers, kernel size, activations, input size are in the following image.

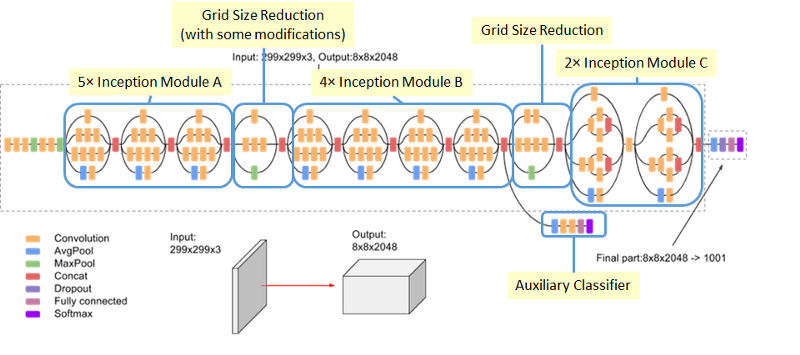




This is my architecture which gave me **6% test accuracy** which is dead low for any system.

So we decide to pursue transfer learning from architectures pretrained on Image Net.

Since our data set is large and have 120 classifications we used inception V3 architecture with weights pre-trained on image net.



Source: <https://medium.com/@sh.tsang/review-inception-v3-1st-runner-up-image-classification-in-ilsvrc-2015-17915421f77c>

The input and output size varies as the portrayed image shows classification of ILSVRC.



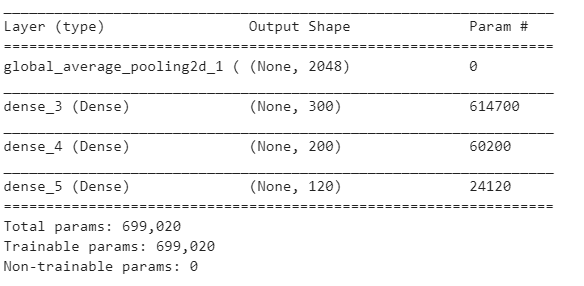
Here are some of layers in inception v3.

We pass our data through the whole network keeping weights constant and extract bottleneck features at mixed 10 layer in inception v3 .chopped of finale layers which are specific to imagenet classification and added A deep neural net for classifying our data set.

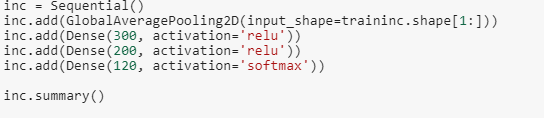
After extracting features

Passed the bottle neck features through following deep neural network

The trainable parameters obtained are 21 million which is too high and gpu memory insufficient to work on it so applied global avg pooling layer to and the parameters have decreased in the range of 6 million.



The following activations and hidden number of nodes were used in training this network.

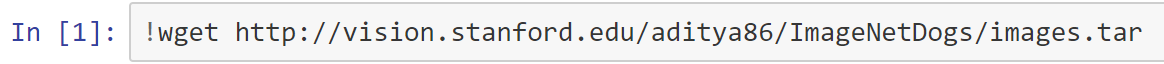
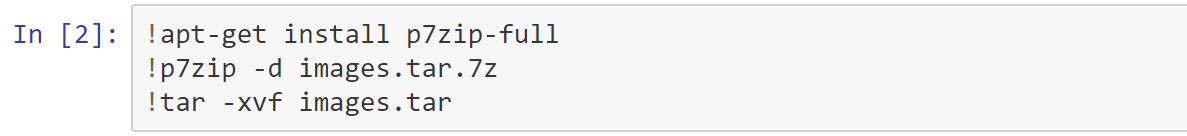


Mostly relu activation is used in DNN architecture and for output we used softmax.

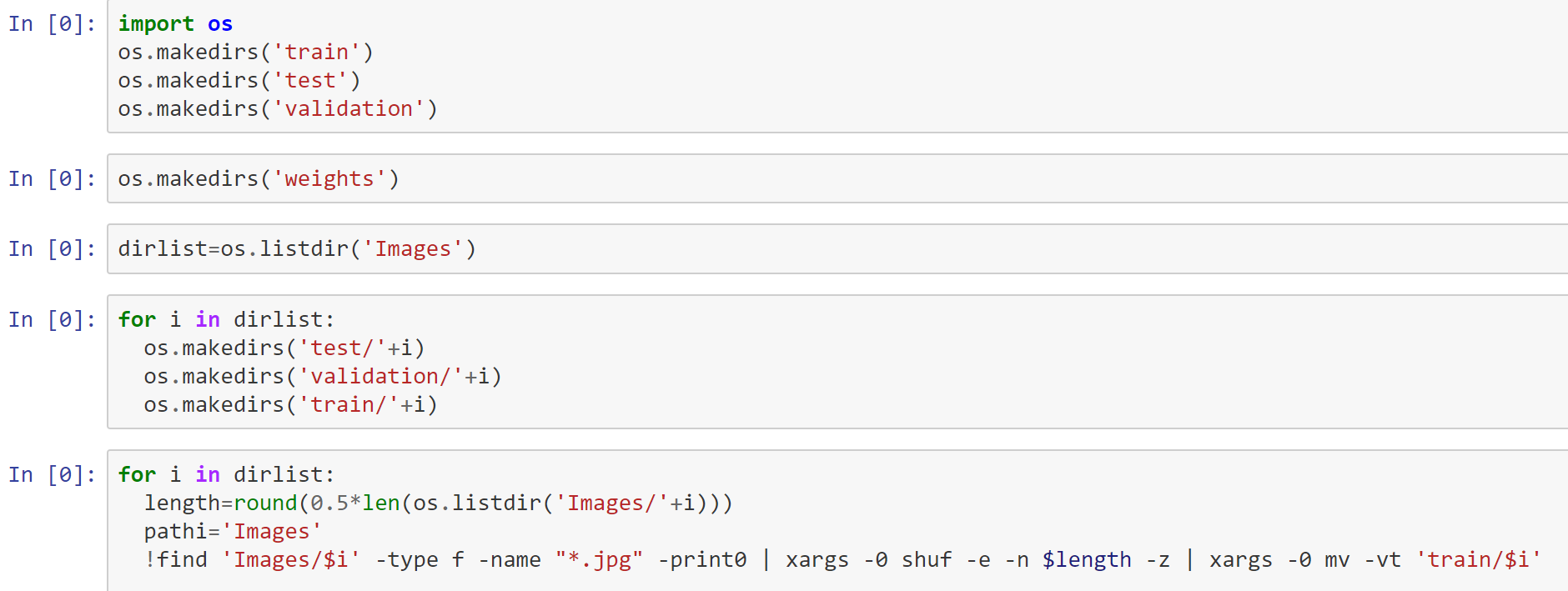
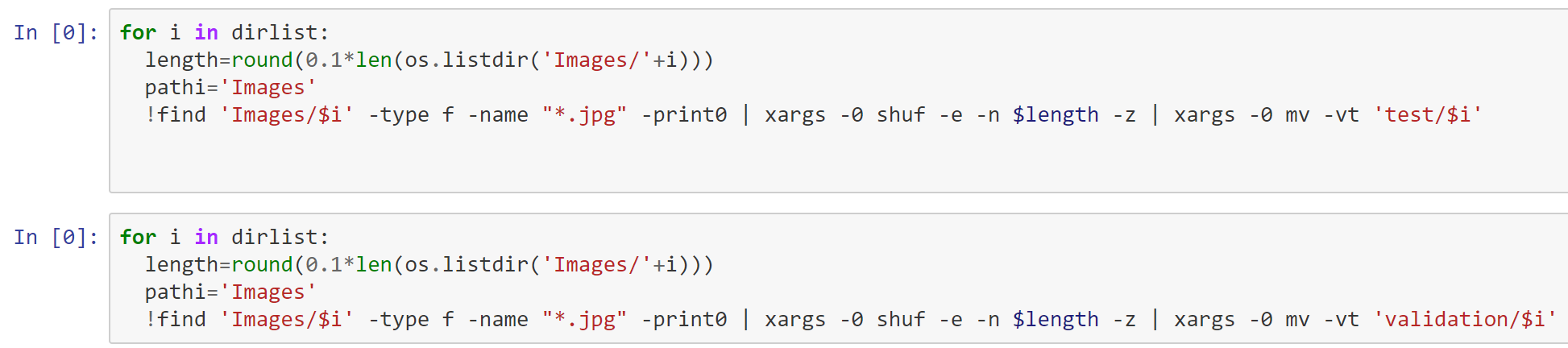
**Methods**

If your project involves other methods describe them in a separate sub-section.

First of all, we need to get those images into python:

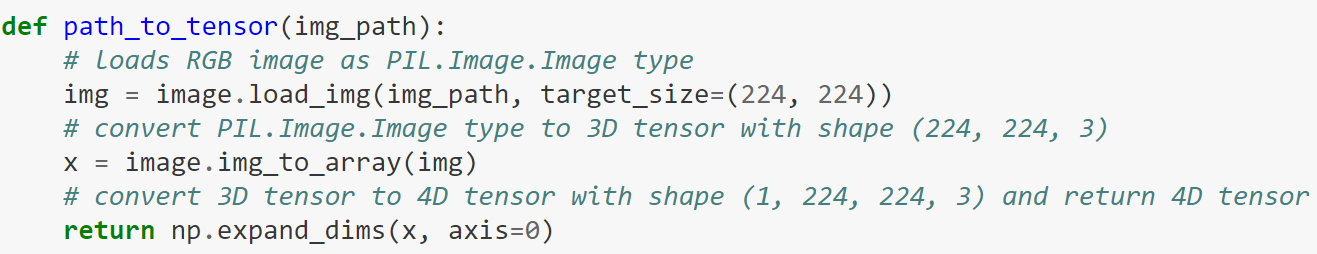
And we set them into three catalogs named “train”, ”test” and “validation” using os library. Then give them name with catalogs.

  
  
similarly, we move the rest of images with different catalogs into test and validation:  
  
  
next, we use several library package in python:

: Load files with categories as subfolder names.  
:Converts a class vector (integers) to binary class matrix.

Keras is a simple modular neural network framework based on the underlying development of Tensorflow, so building a network structure with Keras is simpler than Tensorflow. Here we will use Keras Classifier class provided by Keras, which can be used as Estimator in the scikit-learnpackage, so using this class we can easily call some functions in the sklearnpackage for data preprocessing and result evaluation.

After that we load RGB images as PIL.image.image type and convert this type to 3D tensor with shape (224, 224, 3). Then continue to convert 3D tensor to 4D tensor with shape (1, 224, 224, 3) and return to 4D tensor



This way we reshape all the incoming images into 224\*224, square image size.

Define model structure:

  
we used activation functions as Relu.

We used dropout regularization method.

Every model.add layer has default argument kernel\_initializer='random\_uniform' which initialize weights.

Hyperparameters:

**No of filters**(to increases number of nodes in convolutional layers)

I started with 16 and went up till 512,

Finally came back to 256.

**Size of the filters**(size of patterns to be detected)

2\*2 was kept constant

**Stride**:

Decides how convolutional filter moves on image horizontally/vertically at a time.

If Stride is 1 then then output will also have same height and width

,depth depends on no of filters we set.

If we increase strides then height and width decreases.

For convolution layers I took stride=1, for pooling I took 2 to reduce dimensionality.

**Padding-**

If filter is extending outside image we either can leave the pixels at corners.

(loss of information) –‘VALID’

Or

Padd with zeros so that filter covers all the pixels-‘SAME’.

Ive choose to keep padding to same so that covnet wont miss any regions of image.

**maxpooling**

since convolution layers produce filter stacks of large depths, the parameters used to compute this will go on increasing which causes to overfit.

So by using max pooling we will select a pooling filter where it takes feature maps and gives a weight which is max in that feature map.

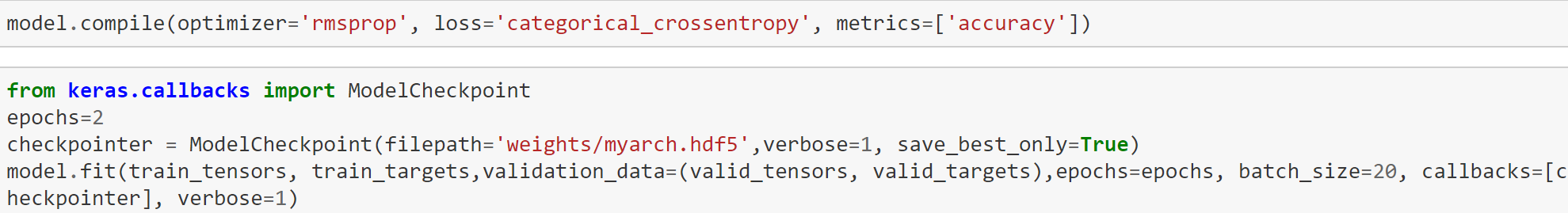
**Some of parameters in pooling:**

• Poolsize:filter size

• Strides:same as pool size, in general we will take it as 2.

• Padding

In the training stage, we use callback function of keras to check the status of the model:



We used cost function as categorical\_crossentropy,

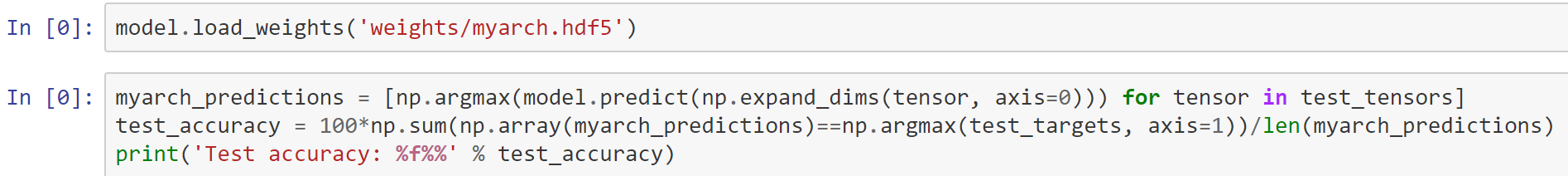
Optimizer as RMS-Propagation.

We are measuring accuracies.

We ran it for 2 epochs with mini batch size =20

And saved the weights.

Test accuracy:



The result of this test is around 5.45% .

Then we tried transfer learning from inceptionv3 pretrained on image net:

In transfer learning, we start with the pre-trained weights for the whole network. Then we fix the weights up to the last layer and let the weights in that layer change as we train on new data.

InceptionV3().predict

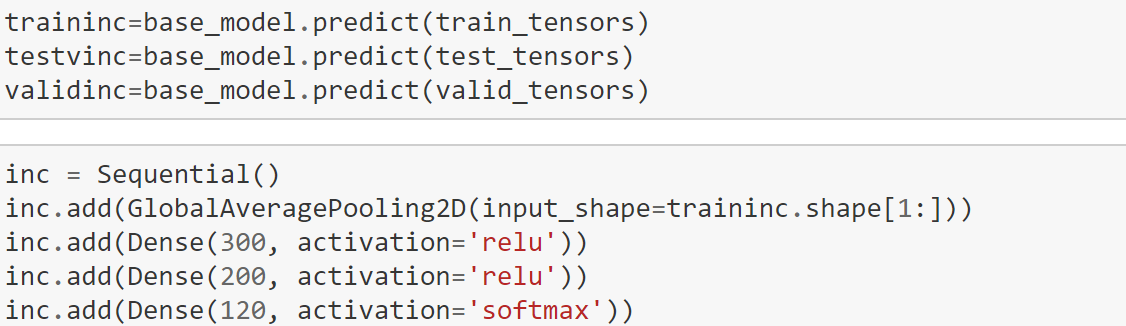
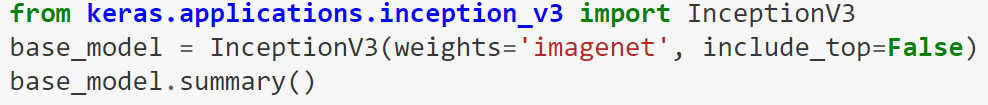
It gives bottle neck features

Since our data is similar to subsets of image net ,

We will initialize inception architecture with imagenet weights.

We will freeze the network remove finale layers and extract bottle neck features.

As shown below:



Imported inception v3 remove finale layers by Include top=false argument.

Using model.predict we got bottle neck features.

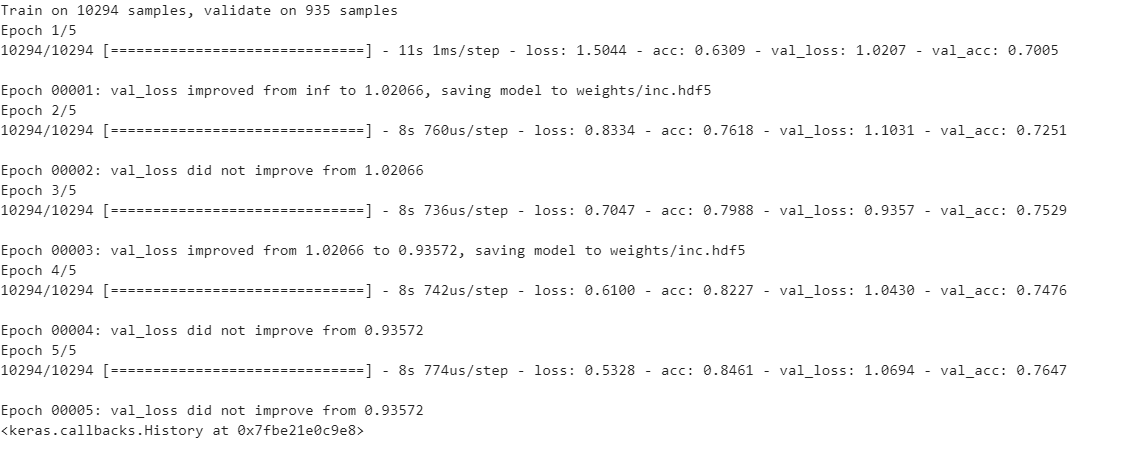
Then we build a neural net with 2 hidden layers as shown above with activations as relu and softmax.



Used cross entropy as loss and rms optimizer

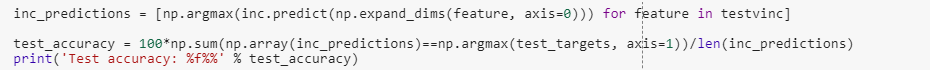


We ran it for 5 epochs , took batch size of 20.



After 3rd epoch our model started over fitting even though train accuracy is increasing validation accuracy started decreasing after 75.29%

So considering that weights we tested our model performance on test set



Got a accuracy of nearly 75%,

Which is a decent accuracy compared to our custom build architecture.

This can accuracy can be improved by data augmentation, trying different architectures like vgg 19, extracting early convolutional layers bottle neck features etc.

Since this is a large data set the gpu memory is getting filled up and kernel is crashing, so we couldn’t apply data augmentation , run more epochs , add more dense layers etc.

So we settled for 75% accuracy.

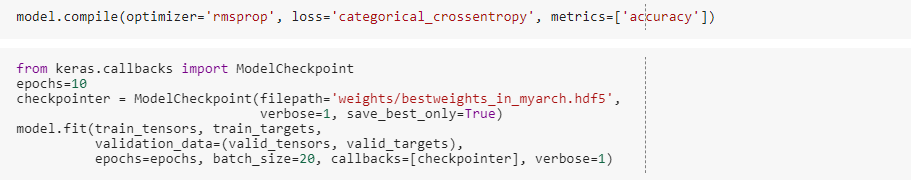
However in order to get proper understanding of data augumentation and other experimenting part we missed, we picked another data set and applied image augumentation as follows. We have worked with this dataset so that we can prove we understand cov nets. gpu capability is what limited us from further experimenting in our previous model.

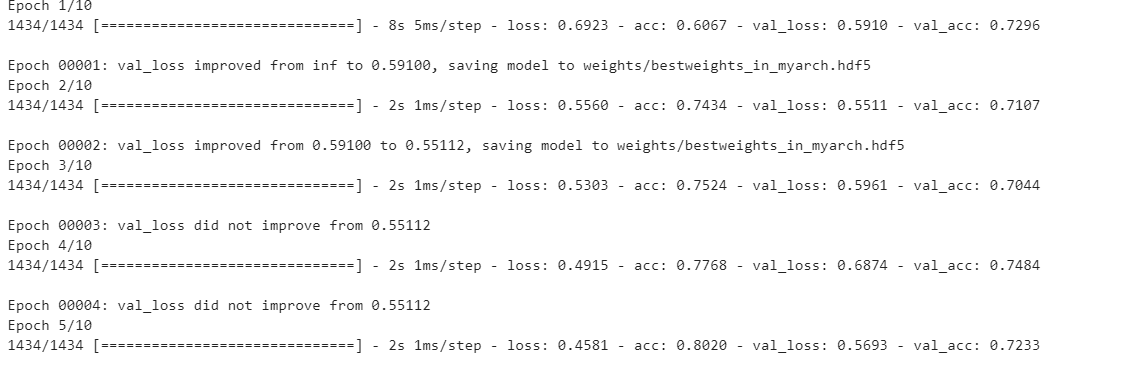
Given the new dataset of 2000 images.

With 1000 images for each artist.

For that we implemented vgg19 model transfer learning.

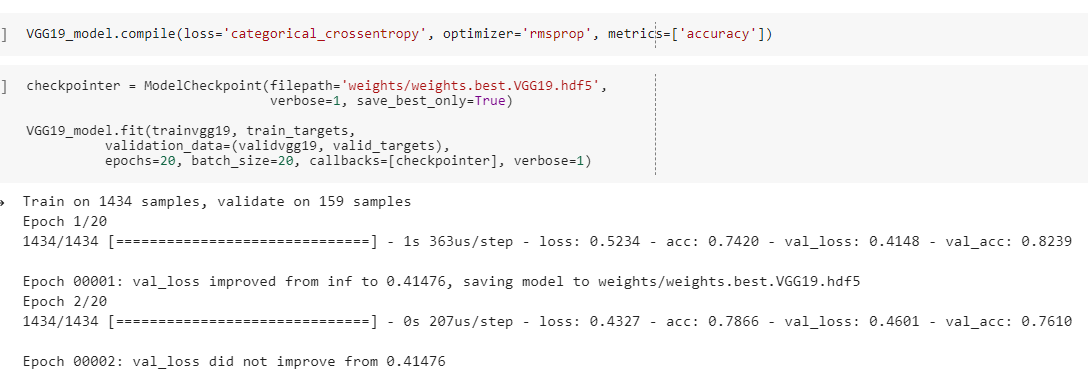


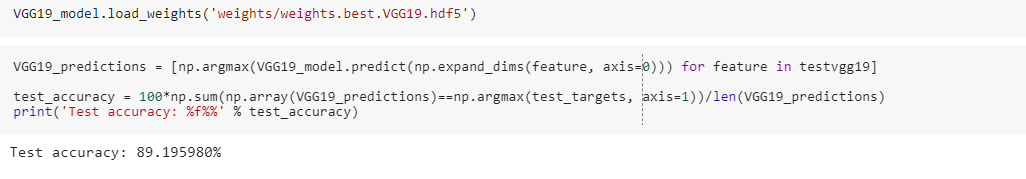




Got an accuracy of around 80%,

  
after using vgg 19 model transfer learning by just chopping finale layers





Got an accuracy of 90%.

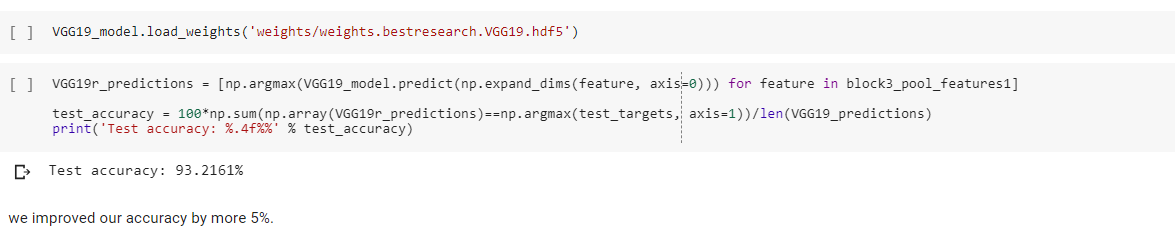
Since this dataset is different from image net, so while initializing weights the end level cov layers learn high level features which wont be useful in our dataset.

So After some experimenting and reading we realized this convolution layers are learning high level features of image net which makes that architectures to poorly perform on different dataset like ours.

So we need to only consider convolution nets with low level features like till ‘block4 pool’ in vgg so that they will not learn more high level features like detecting cars and other objects in ImageNet.

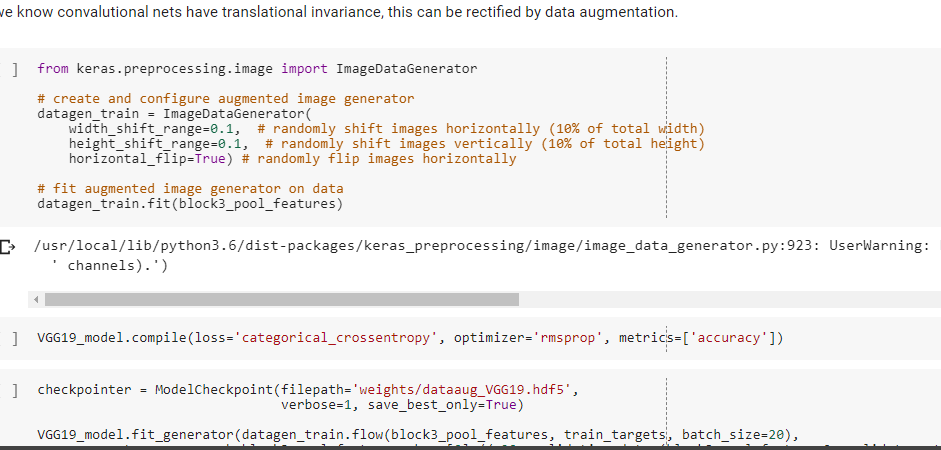
After chopping them off and training our data I got a test accuracy of 93%.

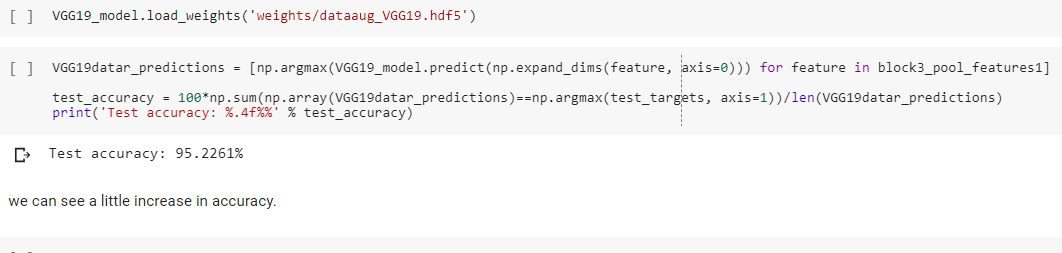




By doing that we got an accuracy of 93%

As we know covnets suffer from translation invariance, and data augmentation can solve it.





we experimented with data augmentation on same model which gave us a finale test accuracy of 95.22%.

**Results:**

**dog dataset:**

**Our Architecture:**

Training accuracy went up to **40%**,

While validation accuracy is **8%.**

This is the highly overfitting model.

Test accuracy of **6%.**

**Transfer Learning:**

**Inception v3:**

Training accuracy went upto **84%**,

validation accuracy is **75%**

test accuracy:**75%.**

**Artist dataset:**

**For trying data augmentation**:

**Our Architecture:**

Training accuracy went up to **90%**,

While validation accuracy is **80%.**

This is the highly overfitting model.

Test accuracy of **80%.**

**Transfer Learning:**

**Vgg19:**

Training accuracy went upto **95%**,

validation accuracy is **90%**

test accuracy:**90%.**

**Vgg19 with data augmentation:**

Training accuracy went upto **98%**,

validation accuracy is **95%**

test accuracy:**95%.**

**Accuracy table.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **Train accuracy** | **Validation accuracy** | **Test accuracy** |
| **Our architecture**  **COV nets+**  **Dense layers** | **40%** | **8%** | **6%** |
| **Inception v3**  **+**  **Our Deep neural net** | **84%** | **75%** | **75%** |

**Comments:**

**Our architecture is not deep enough to extract convolution filters capable of identifying breeds.**

**It is highly overfitting.**

**Passing the data through whole network and training needs a gpu with more memory.**

**So it is not a feasible plan for us to train the entire convalution network.**

**So we took inception v3 with weights of imagenet.**

**Passed the bottle neck features and were able to achieve 75%.**

**We know we can further increase accuracy by increasing dense layers,data augmentation etc, but with our dataset of 20,000 high resolution images ,our kernel will instantly crash.**

**So we took another dataset and done same as above and tried data aug and other transfer learning techniques.**

**Dataset is 2000 images of different art.**

**Results table.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **Train accuracy** | **Validation accuracy** | **Test accuracy** |
| **Our architecture**  **COV nets+**  **Dense layers** | **90%** | **80%** | **80%** |
| **Vgg19(chopped finale dense layers)**  **+**  **Our Deep neural net** | **92%** | **85%** | **85%** |
| **Vgg19(chopped finale convolution layers and dense and extracted bottleneck features at previous pooling layers i.e )**  **+**  **Our Deep neural net** | **96%** | **92%** | **92%** |
| **Above architecture with data augumentation** | **98%** | **95%** | **95%** |

**\*All the accuracies are rounded off.**



We downloaded images from internet and gave it to our system.

**References (5 points)**

Deep learning course ppt at UC(spring 2019)

Intelligent systems course ppt at UC(fall 2018)

<https://www.kaggle.com/gaborfodor/dog-breed-pretrained-keras-models-lb-0-3>

<https://www.kaggle.com/orangutan/keras-vgg19-starter>

<https://www.kaggle.com/twhitehurst3/stanford-dogs-keras-vgg16>

<https://www.kaggle.com/stassl/displaying-inline-images-in-pandas-dataframe>

<https://www.kaggle.com/methindor/dogbreeddatavisualisation>

<https://www.kaggle.com/jcesquiveld/transfer-learning-for-dog-breed-classification-ii>

DeepLearning.ai Deep Learning course

<https://www.kdnuggets.com/2017/12/getting-started-tensorflow.html>

<https://towardsdatascience.com/dog-breed-classification-hands-on-approach-b5e4f88c333e>

**\*tittle photo from google images**