Capstone Project

Abhishek Ravindran

Machine Learning Engineer Nanodegree

Feb 12, 2019

I. Definition

Project Overview

After centuries of intense whaling, recovering whale populations still have a hard time adapting to warming oceans and struggle to compete every day with the industrial fishing industry for food. To aid whale conservation efforts, scientists use photo surveillance systems to monitor ocean activity. They use the shape of whales' tails and unique markings found in footage to identify what species of whale they're analysing and meticulously log whale pod dynamics and movements. For the past 40 years, most of this work has been done manually by individual scientists, leaving a huge trove of data untapped and underutilized.

For example, the organization B.C. Cetacean Sightings Network uses sightings to help create safer waters for cetaceans and sea turtles. Information about the distribution and abundance of species is the key to both understanding the status of the species (extinct, extirpated, endangered, threatened, at-risk or not at risk – are the COSEWIC categories), and whether the status of the population is improving, steady or declining. Once any particular endangered/threatened whale species has been spotted the WhaleReport Alert System (WRAS) broadcasts pertinent details of whale presence to bridge crew and pilots of large commercial vessels so that they can implement speed reductions or other measures to reduce their risk of disturbing or striking whales. Identifying the whales and tagging it to the exact species can be challenging.

So this can be accomplished using the power of deep learning algorithm.

Problem Statement

An algorithm to identify individual whales by the images of their tail has to be developed. We'll analyse the Happywhale's database of over 25000 images gathered from research institutes and public contributors. The algorithm will predict the probability of an image being of a particular whale breed. Maximum of 4/5 whale suggestion can be given for any of the image with the corresponding probability likelihood.

Convolutional neural network seems to be a viable option. CNN is a special architecture of artificial neural networks, proposed by Yann LeCun in 1988. CNN uses some features of the visual cortex. One of the most popular uses of this architecture is image classification. For example Facebook uses CNN for automatic tagging algorithms, Amazon — for generating product recommendations and Google — for search through among users' photos. The power of this network can be used to identify the whale species based on their tails. Each of the tail will be different in some way or the other, like some may have a crack on the end others vary in colour. So these are the features that are looked more closely by the network and predicts the probability of the whale being a particular species.

Kaggle link: https://www.kaggle.com/c/humpback-whale-identification/data

Metrics

Submissions are evaluated according to the Mean Average Precision.

$$MAP@5 = \frac{1}{U} \sum_{u=1}^{U} \sum_{k=1}^{min(n,5)} P(k)$$

where U is the number of images, P(k) is the precision at cutoff k, and n is the number predictions per image. MAP@5 means 5 items are recommended for each user. So to calculate the Mean average precision we need to know the below:

1. We can recommend at most 5 items for each user.
2. It pays to submit all 5 recommendations, since we are not penalized for bad guesses.
3. Order matters, so it is better to submit more certain recommendations first, followed by the recommendations we are less sure about.

The reason to choose this metric is because predicting the whale species exactly from more than 5000+ species is like trying to pick a needle out of a haystack. So in this scenario we will calculate top 5 of the possible species for any particular whale. That said we would pick top 5 highest probability whale species.

II. Analysis

Data Exploration

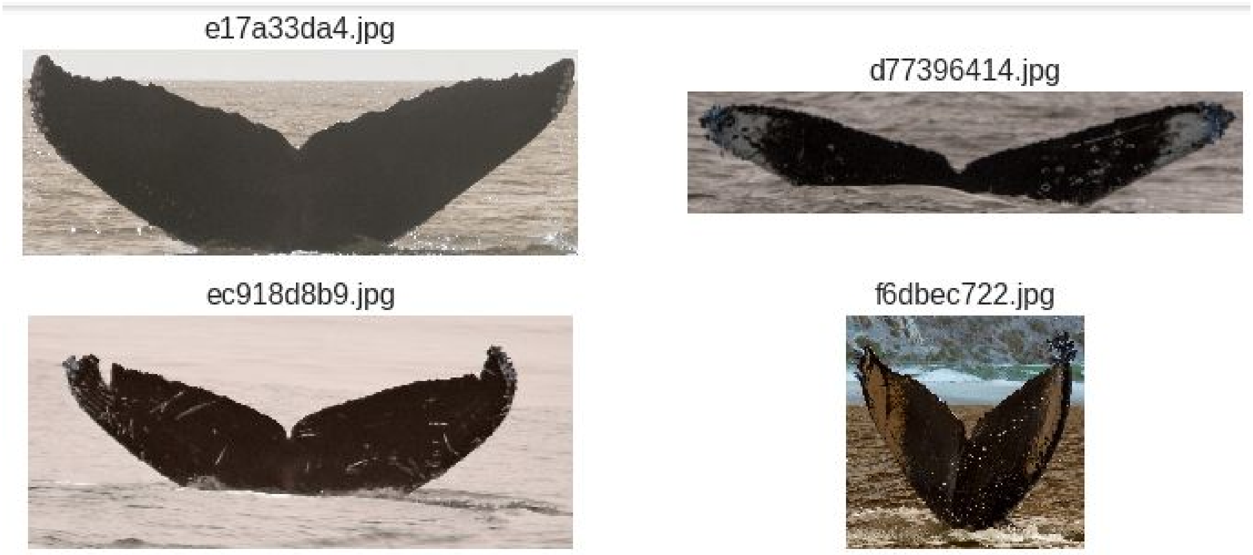
The dataset consist of images of different whale's tail obtained from the Kaggle competition "Humpback whale identification". The training data consists of thousands of images of humpback whale flukes. Individual whales have been identified by researchers and given an Id. There are almost 3000+ whale ids.

The following are the file description and their corresponding download link:

- train.zip – This zip folder contains the training images. The model will be trained using the same.

- train.csv – This CSV file comprises the mapping between the images and the whale id (label). Whales that are not predicted to have a label identified in the training data will be labelled as ‘new_whale’.
- test.zip – This zip folder consists of test images which the model would use to predict its corresponding label.
- sample_submission.csv – A sample submission file with the corresponding image file name and its Id.

Sample Image:



The training data has a few unlabelled whale images as well; these are denoted by ‘**new_whale**’ in the CSV. These would be removed before the training is done.

Below are the hyperlinks to the dataset (publically available):

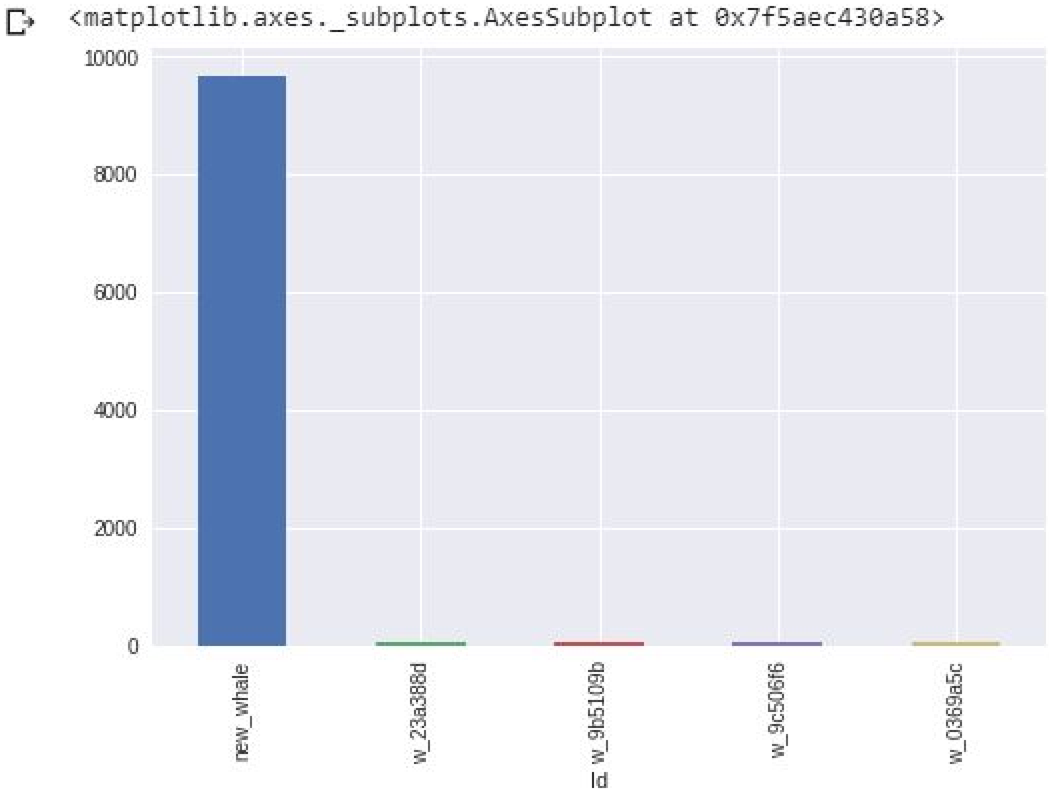
https://www.kaggle.com/c/humpback-whale-identification/download/train.zip

https://www.kaggle.com/c/humpback-whale-identification/download/train.csv

https://www.kaggle.com/c/humpback-whale-identification/download/test.zip

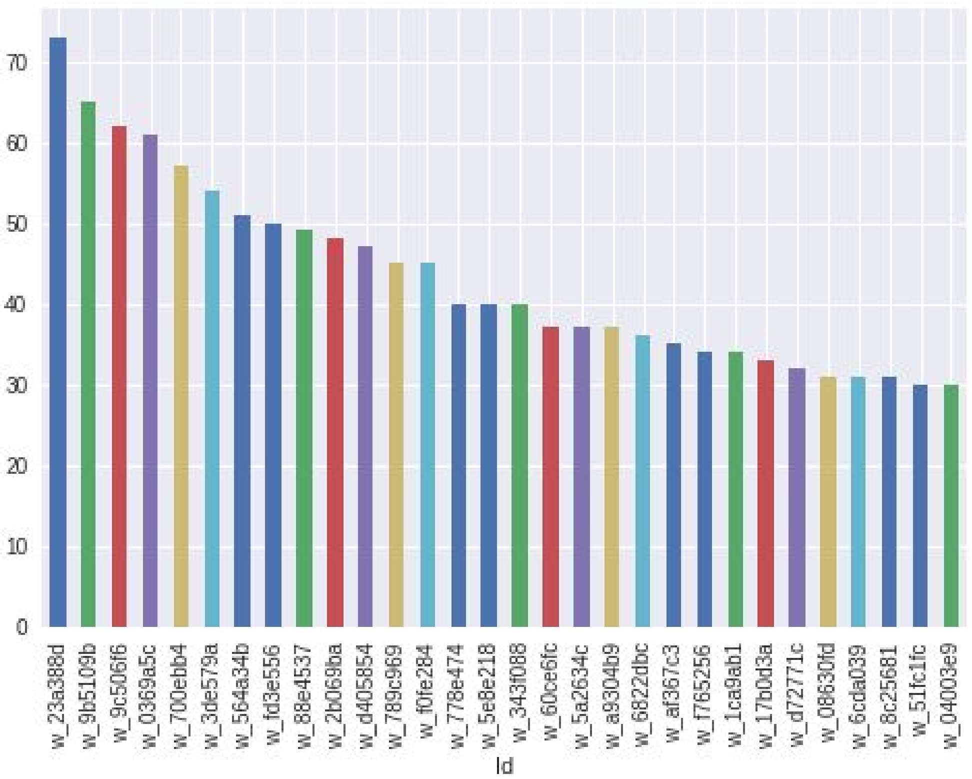
Exploratory Visualization

The number of samples with ‘new_whale’ is around 10k, which means that these many images are not pre-classified. This can be seen from the below illustration.

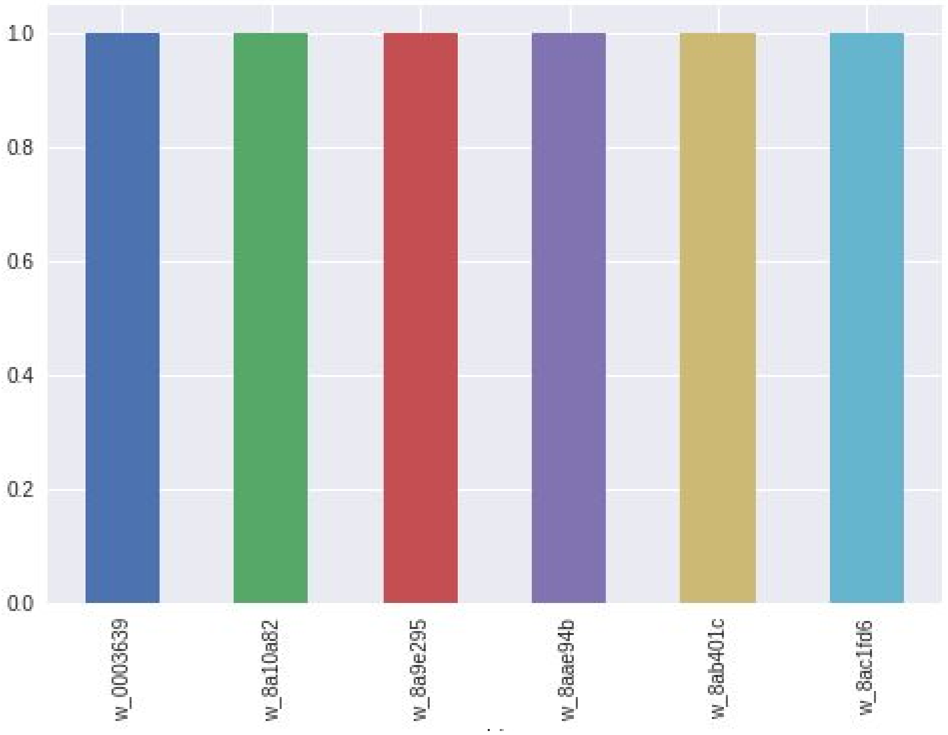


```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5aec430a58>
```

As a result, ‘new_whale’ label and its corresponding samples need to be removed before training. Once removed the graph of top 30 labels can be seen below.



It can be seen that there are entries varying from more than 70 samples per label to 1 sample per label.



Algorithms and Techniques

The algorithm that will be used is Convolutional Neural Network (CNN), which is a class of deep learning algorithm which is widely used for image classification. CNN, like neural networks, are made up of neurons with learnable weights and biases. Each neuron receives several inputs, takes a weighted sum over them, passes it through an activation function and responds with an output. The whole network also has a loss function. CNN along with transfer learning is a deadly combination and this is the same that is used in this prediction model as well. A VGG16 model that already had been trained quite a lot on millions of images is being reused along with few convolutional/dense layer and max pooling layer at the end. This is done to ensure that the model is tailor-made for whale species prediction and nothing else. A **'relu'** activation function is used along with the additional convolutional/dense layer that's updated later on in the network. The final dense layer would have the total number of labels as an output along with **'softmax'** activation function to predict the probability of it being a particular species of whale.

The input fist is preprocessed by reshaping it to (160,120). Since most of the whale labels are having only 1 or 2 samples, more samples need to be recreated for training. For this we shall rely on image augmentation to rotate, sheer, zoom, flip the existing image to get more samples. Apart from this 'sgd' optimizer was used.

Benchmark

Benchmark of the model is to get an accuracy of 50% or higher when predicting. This particular benchmark is chosen as it is not easy to choose from 5000+ whale labels. A random guess would give a probability of 0.02%.

III. Methodology

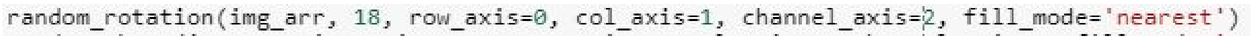
Data Preprocessing

Initially, all the training images were stored in Google drive, which was then being retrieved based on the train.csv file. On closer analysis as seen in the data exploration step, there are a lot of labels with very few samples, the majority of the images had only 1 sample. So to deal with this issue each of these images had to be augmented in such a way that more samples were produced. Each of these newly produced samples was different from each other in a way that few were stretched, zoomed into, flipped horizontally or rotated a bit.

Keras library has a preprocessing image function that has random rotations, random sheer, shift and so on.

Random rotation:

Sometimes images in the sample data may have varying and different rotations in the scene. Training the model to better handle rotations of images is done by artificially and randomly rotating images from the dataset.

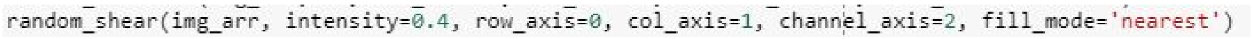


```
random_rotation(img_arr, 18, row_axis=0, col_axis=1, channel_axis=2, fill_mode='nearest')
```

The rotation range of 18 degrees is given so that the image would be turned a bit 18 degree sideways.

Random shear:

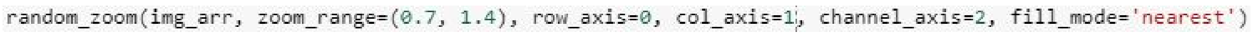
A random spatial shear of the numpy image tensor is also a part of the augmentation pipeline. A transformation intensity of 0.4 in deg is given as a parameter to this function.



```
random_shear(img_arr, intensity=0.4, row_axis=0, col_axis=1, channel_axis=2, fill_mode='nearest')
```

Random zoom:

This function is intended to perform a random spatial zoom. A zoom range in tuple needs to be specified. (0.7, 1.4) were the ranges that were specified.

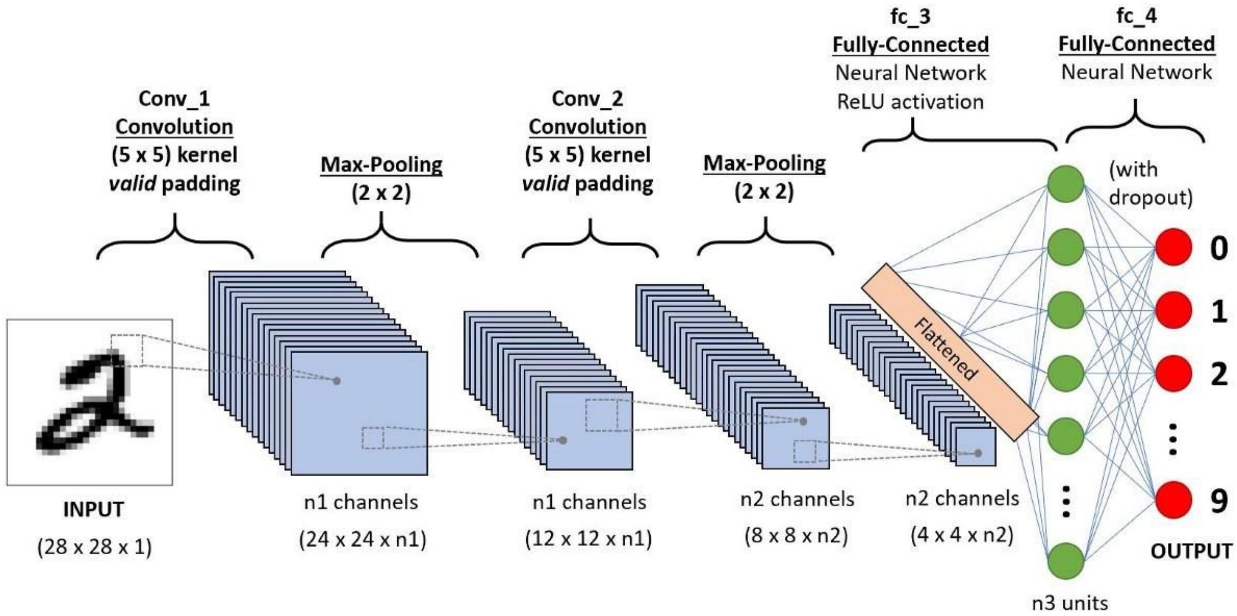


```
random_zoom(img_arr, zoom_range=(0.7, 1.4), row_axis=0, col_axis=1, channel_axis=2, fill_mode='nearest')
```

Implementation

When coming to the implementation phase convolutional neural network (CNN) was used along with transfer learning to create the model.

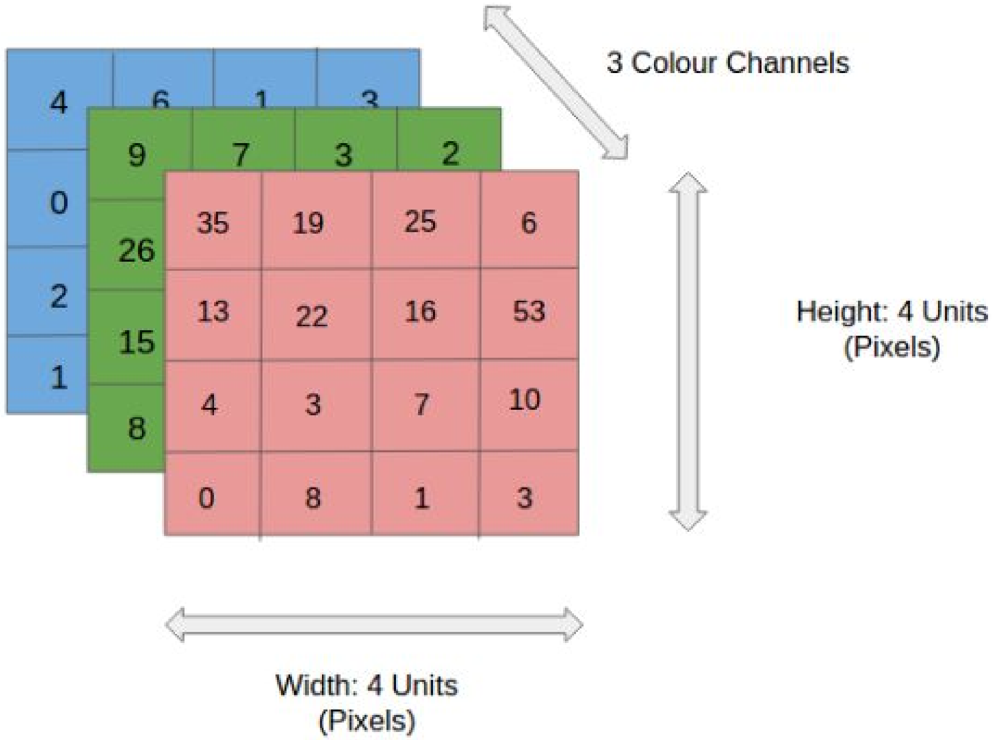
A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.



The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area.

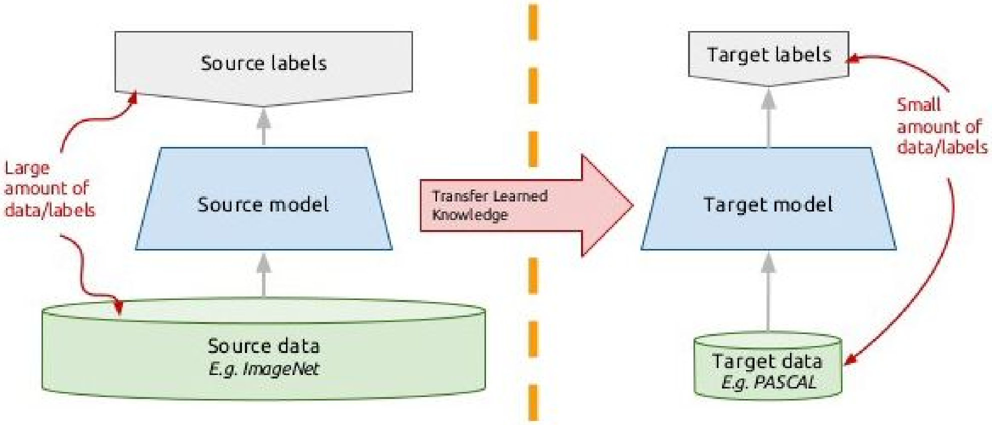
Advantage of CNN over MLP:

A ConvNet is able to successfully capture the Spatial and Temporal dependencies in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights. In other words, the network can be trained to understand the sophistication of the image better.

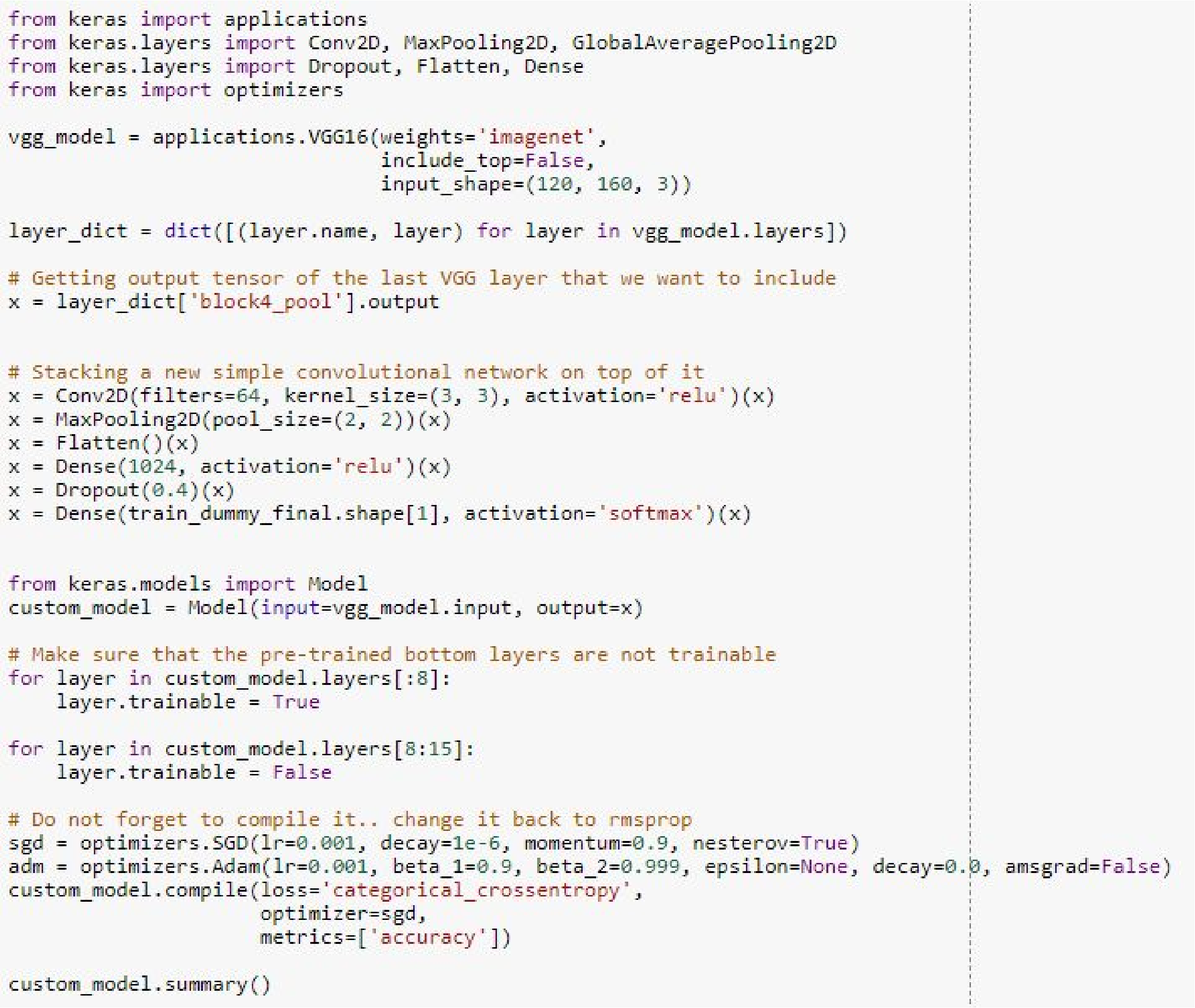


In the figure, we have an RGB image which has been separated by its three color planes — Red, Green, and Blue. There are a number of such color spaces in which images exist — Grayscale, RGB, HSV, CMYK, etc. We can imagine how computationally intensive things would get once the images reach dimensions, say 8K (7680×4320). The role of the ConvNet is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction. This is important when we are to design an architecture which is not only good at learning features but also is scalable to massive datasets.

Using transfer learning:



The basic premise of transfer learning is simple: take a model trained on a large dataset and transfer its knowledge to a smaller dataset. For object recognition with a CNN, we freeze the early convolutional layers of the network and only train the last few layers which make a prediction. The idea is the convolutional layers extract general, low-level features that are applicable across images — such as edges, patterns, gradients — and the later layers identify specific features within an image such as eyes or wheels.



```
from keras import applications
from keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D
from keras.layers import Dropout, Flatten, Dense
from keras import optimizers

vgg_model = applications.VGG16(weights='imagenet',
                               include_top=False,
                               input_shape=(120, 160, 3))

layer_dict = dict([(layer.name, layer) for layer in vgg_model.layers])

# Getting output tensor of the last VGG layer that we want to include
x = layer_dict['block4_pool'].output


# Stacking a new simple convolutional network on top of it
x = Conv2D(filters=64, kernel_size=(3, 3), activation='relu')(x)
x = MaxPooling2D(pool_size=(2, 2))(x)
x = Flatten()(x)
x = Dense(1024, activation='relu')(x)
x = Dropout(0.4)(x)
x = Dense(train_dummy_final.shape[1], activation='softmax')(x)


from keras.models import Model
custom_model = Model(input=vgg_model.input, output=x)

# Make sure that the pre-trained bottom layers are not trainable
for layer in custom_model.layers[:8]:
    layer.trainable = True

for layer in custom_model.layers[8:15]:
    layer.trainable = False

# Do not forget to compile it.. change it back to rmsprop
sgd = optimizers.SGD(lr=0.001, decay=1e-6, momentum=0.9, nesterov=True)
adm = optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False)
custom_model.compile(loss='categorical_crossentropy',
                     optimizer=sgd,
                     metrics=['accuracy'])

custom_model.summary()
```

In this whale identification project, the VGG16 model which was pre-trained over millions of ImageNet data is being reused. The final set of extra layers are added which would tailor make the entire model to meet our needs. Since the data per sample image is quite less the initial few layers are made trainable (initial 8) and the layers from 8 to 15 are made not trainable.

The reason for making the initial few layers are trainable is because of the dataset we have. It belongs to the small and different data when compared to the original dataset the VGG16 model was trained on. When using transfer learning approach there can be four main cases that we can come across:

1. New data set is small; new data set is similar to original data set.
2. New data set is small; new data set is different from the original data set.
3. New data set is large; new data set is similar to original data set.

4. New data set is large; new data set is different from the original data set.

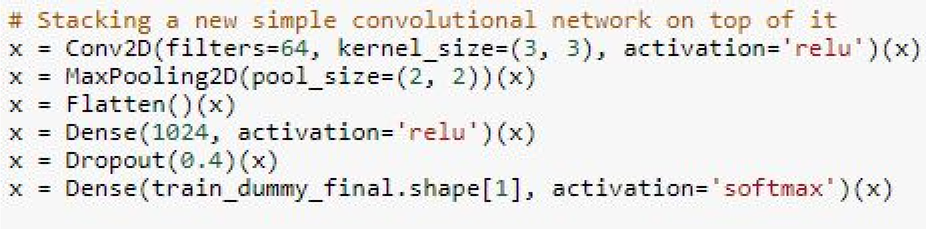
This particular data set belongs to the second category. So if that is the case, we need to:

1. Slice off most of the pre-trained layer's near the beginning of the network.
2. Add to the remaining pre-trained layers a new fully connected layer that matches the classes in the new data set.
3. Randomize the weights of the new fully connected layer; freeze all the weights from the pre trained network.
4. Train the network to update the weights of the new fully connected layer.

This will ensure that the new model that we created is tailor made for our dataset.

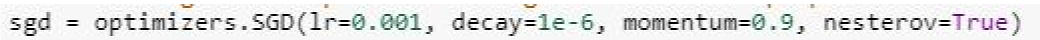
Refinement

The newly added CNN layer after the transfer layer has 64 filters with kernal size of (3, 3) and it also has a 'relu' activation function. This ensures that the positive value remains the same and the negative becomes zero. A max pooling layer is added after this CNN layer and it has a pool size of (2, 2). Dense layer with 1024 along with a 'relu' activation function comes after the pooling layer. Finally a dense output layer with an activation function of 'softmax' is also added to get the probability of the output values.



```
# Stacking a new simple convolutional network on top of it
x = Conv2D(filters=64, kernel_size=(3, 3), activation='relu')(x)
x = MaxPooling2D(pool_size=(2, 2))(x)
x = Flatten()(x)
x = Dense(1024, activation='relu')(x)
x = Dropout(0.4)(x)
x = Dense(train_dummy_final.shape[1], activation='softmax')(x)
```

Initially, stochastic gradient descent was used as the optimizer with default parameters and I wasn't able to get much accuracy on that. So had to modify the optimizer to have a learning rate of 0.001 and tweak the other parameters as shown below.



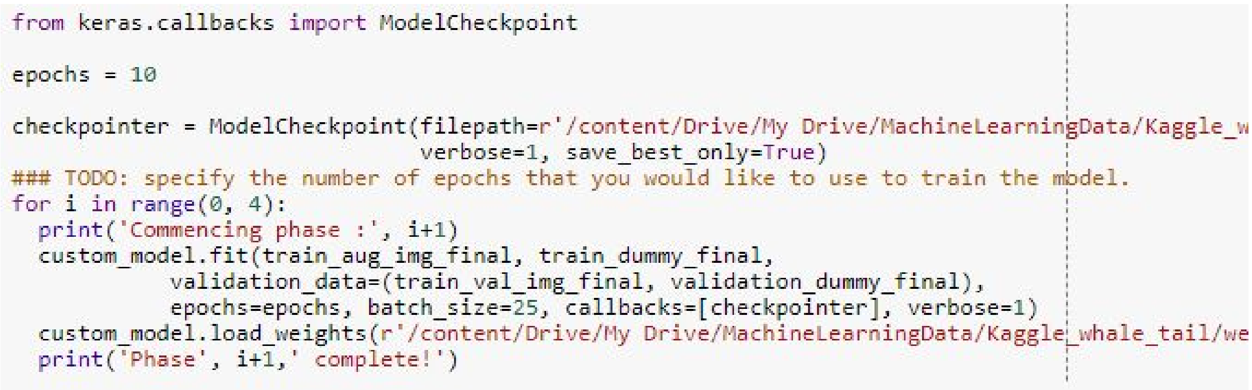
```
sgd = optimizers.SGD(lr=0.001, decay=1e-6, momentum=0.9, nesterov=True)
```

This improved the accuracy a lot more than initially observed. And at last the model was trained over 10 epochs with a batch size of 25. A callback checkpoint was also used to save the best weights onto a file.

IV. Results

Model Evaluation and Validation

The model has a loss function of cross entropy and accuracy as its metrics. The optimizer used is the stochastic gradient descent. Model checkpoint is imported from keras.callbacks to ensure that the best weights are saved as the training progresses.

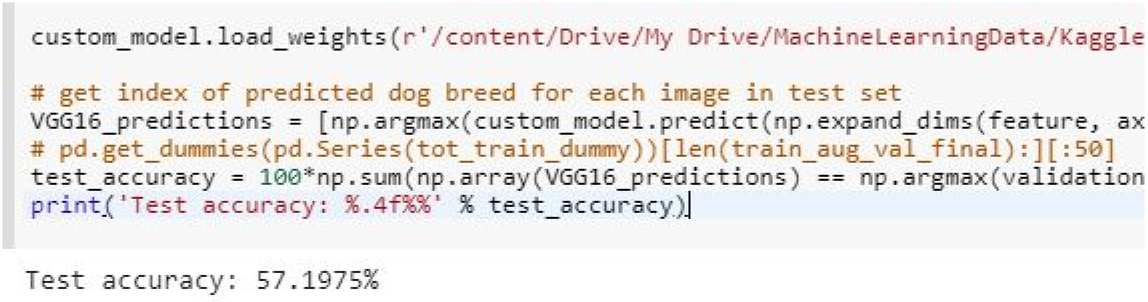


```
from keras.callbacks import ModelCheckpoint

epochs = 10

checkpointer = ModelCheckpoint(filepath=r'/content/Drive/My Drive/MachineLearningData/Kaggle_w
                               verbose=1, save_best_only=True)
### TODO: specify the number of epochs that you would like to use to train the model.
for i in range(0, 4):
  print('Commencing phase :', i+1)
  custom_model.fit(train_aug_img_final, train_dummy_final,
            validation_data=(train_val_img_final, validation_dummy_final),
            epochs=epochs, batch_size=25, callbacks=[checkpointer], verbose=1)
  custom_model.load_weights(r'/content/Drive/My Drive/MachineLearningData/Kaggle_whale_tail/we
  print('Phase', i+1,' complete!')
```

The whole data is split into the train and validation set. The training batch size is 25 and the model has an epoch count of 10. And as the validation loss becomes lesser compared to the previous epoch the weights of the same gets saved in the file. Initially, the input data was having a lot of variations, like few of the labels had more samples than the others. On training the model with such a dataset I was able to notice that the prediction accuracy was not as good on seeing a new dataset. This changed after I ensured that there was an equal distribution of data for each of the samples.

Justification

Once the training is done on using the unseen data to do prediction I was able to get a **57%** accuracy. This is more than the benchmark score that was set by me previously. Considering that there are more than 5000 labels a random guess would yield an accuracy of 0.02 % which is extremely low. Apart from this I have also calculated the accuracy based on the first 5 best-suggested labels and was able to notice that I am able to get a 78% accuracy.



```
custom_model.load_weights(r'/content/Drive/My Drive/MachineLearningData/Kaggle

# get index of predicted dog breed for each image in test set
VGG16_predictions = [np.argmax(custom_model.predict(np.expand_dims(feature, ax
# pd.get_dummies(pd.Series(tot_train_dummy))[len(train_aug_val_final):][:50]
test_accuracy = 100*np.sum(np.array(VGG16_predictions) == np.argmax(validation
print('Test accuracy: %.4f%%' % test_accuracy)
```

```
Test accuracy: 57.1975%
```

V. Conclusion

Free-Form Visualization

By just having a glance at the images it can be seen that there are some qualities about each of the image that differentiates the whale tails. Maybe a speck of dot or a crack at the end of the tail and if the model can get a hold of these qualities it would make it much easier to predict the whale species from just the tail.

Reflection

On reflecting back on the whole problem there were quite a few challenges I had to face, a major one being the unequally distributed dataset. Meaning there were a lot of labels which had only 1 sample. So this called for data augmentation, where the image was either randomly rotated, sheared, flipped etc. This was a necessary step since the training would have been biased towards labels which had more samples in it.

Apart from this, another challenge came up during training where the parameter tweaking is an important part. I had to get the learning rate, momentum and other parameters just about right to get better accuracy. Initially sgd along with default parameters were used, which did not improve the accuracy much, but later on on setting the below parameters a better accuracy was achieved.

- learning rate - 0.001

- decay rate - 1e-6
- momentum - 0.9
- nesterov - True

Improvement

I would probably try out quite a lot of optimizer out there with various parameters to see if the accuracy would improve. I am certain that if I get more images samples for the labels that had only 1 or 2 samples; that could help training the algorithm much better. This would obviously improve accuracy.