MEF UNIVERSITY

HUMPBACK WHALE IDENTIFICATION WITH CONVOLUTIONAL NEURAL NETWORKS

Capstone Project

Duygu Can

İSTANBUL, 2018

MEF UNIVERSITY

THE HUMPBACK WHALE IDENTIFICATION WITH CONVOLUTIONAL NEURAL NETWORKS

Capstone Project

Duygu Can

Advisor: Assoc. Prof. Şuayb S. Arslan

İSTANBUL, 2018

MEF UNIVERSITY

Name of the project: Humpback Whale Identification with CNN
Name/Last Name of the Student: Duygu Can
Date of Thesis Defense: 28/12/2018

I hereby state that the graduation project prepared by Duygu Can has been completed under my supervision. I accept this work as a "Graduation Project".

28/12/2018
Assoc. Prof. Şuayb S. Arslan

I hereby state that I have examined this graduation project by Duygu Can which is accepted by his supervisor. This work is acceptable as a graduation project and the student is eligible to take the graduation project examination.

28/12/2018

Director
of
Big Data Analytics Program

We hereby state that we have held the graduation examination of Duygu Can and agree that the student has satisfied all requirements.

THE EXAMINATION COMMITTEE

Committee Member Signature

1. Assoc. Prof. Şuayb S. Arslan

2. Prof. Özgür Özlük

Academic Honesty Pledge

I promise not to collaborate with anyone, not to seek or accept any outside help, and not to give any help to others.

I understand that all resources in print or on the web must be explicitly cited.

In keeping with MEF University's ideals, I pledge that this work is my own and that I have neither given nor received inappropriate assistance in preparing it.

Duygu Can 28.12.2018

EXECUTIVE SUMMARY

THE HUMPBACK WHALE IDENTIFICATION WITH CONVOLUTIONAL NEURAL NETWORKS

Duygu Can

Advisor: Assoc. Prof. Şuayb S. Arslan

DECEMBER 2018, 68 pages

The migration patterns of humpback whales are tracked with conventional photo-identification techniques for decades. The distinct markings on whale flukes serve as unique fingerprints for these creatures. This study aims to identify humpback whales according to their fluke images using ResNET, a deep neural network architecture to help the conservation efforts for this endangered species by automatizing the process. We experimented with different train/test split schemes and initializations to obtain the best classifying model. Although we were limited with a small sized training set of 200 images, using state-of-the-art image processing and data augmentation methods we obtained a high accuracy of 0.94 for 11 distinct whales. This project is served as an friendly interface to dive deep into the field of image recognition with Convolutional Neural Networks.

Key Words: Convolutional Neural Networks (CNN), Humpback Whale, Artificial Neural Networks (ANN), Image Classification, Photo-identification, ResNET.

ÖZET

EVRİŞİMLİ SİNİR AĞLARI KULLANARAK KAMBUR BALİNA TANIMA

Duygu Can

Tez Danışmanı: Doç. Dr. Şuayb S. Arslan

ARALIK 2018, 68 sayfa

Kambur balinaların göç kalıpları, onlarca yıldır geleneksel fotoğraf tanımlama teknikleri ile izlenmektedir. Balina kuyrukları üzerindeki belirgin işaretler, bu canlılar için özgün parmak izleri gibi davranmaktadır. Tanıma sürecini otomatikleştirerek, nesli tehlikede olan kambur balinaların korunma çabalarına katkıda bulunmayı hedefleyen bu çalışmada, bir derin sinir ağı mimarisi olan ResNET ile kuyruk görsellerine göre balina tanıma hedeflenmiştir. En iyi sınıflandırma modelini elde etmek için farklı eğitim/test ayrımı şemaları ve değişik başlangıç noktalari ile deneyler yapılmıştır. 200 görüntüden oluşan küçük bir eğitim seti ile sınırlı kalınmasına rağmen, ileri görüntü işleme ve veri artırma yöntemlerini kullanılarak 11 farklı balina için 0.94 yüksek başarımı edilebilmiştir. Bu proje, Konvolüsyonel Sinir Ağları ile görüntü tanıma alanına derinlemesine dalmak için dostça bir arayüz olarak hizmet etmiştir.

Anahtar Kelimeler: Evrişimli Sinir Ağları (ESA), Kambur Balina, Yapay Sinir Ağları (YSA), Görüntü Sınıflandırma, Foto-tanımlama, ResNET.

TABLE OF CONTENTS

TABLE OF FIGURES

1. INTRODUCTION

After years of intense whaling, large whale species became extinct. Starting from the ancient times, 90% of the humpback whales are massacred during the whaling era (Anonymous, 2018). By 1986, International Whaling Commission banned commercial whaling for all whale species due to severe risk of extinction (International Whaling Commission, 2018). The moratorium is still valid today, but Japan, Norway and Iceland oppose the moratorium decision and establish their own hunting limits. Thanks to the ban, the humpback whale population is recovered and now it is estimated to be at least 80000. Now whaling no longer threatens the whales but other reasons such as collisions with ships, industrial fishing, ocean pollution, noise pollution and global ocean warming are still possible causes of risk for their survival. They travel 25750 kilometers on average on yearly basis for mating and feeding (Anonymous, Whale Facts: Marine Mammal Facts & Information, 2018). For conservation purposes, marine biologists track their migration routes and they get help from individual whale photos for these sacred efforts.

1.1. Brief Literature Review

Starting from the 1970s, scientists use natural markings to recognize an individual of a species (Fuhr, et al., 2016). Especially it is easy to identify the humpback whales because they swim near the surface and when they dive deep for food, they raise their flukes up in the air. On those flukes, there are color variations from white to black, toothmarks of killer whales, algal films and various other scars and scratches. These distinct markings are fingerprints of the individual whales. Change in the color of the whale's skin first noted by Lillie in 1915 but the uniqueness of this variation is not recognized (Lillie, 1915). Years later in 1960, utilizing these colors and marks Schevill and Backus could be able to track the same humpback whale over ten days of journey near Portland, Maine (Schevill & Backus, 1960). Thanks to Kraus and Katona, the first catalogue of 120 fluke photos is formed and later Katona and his colleagues enlarged the list to 1000 individuals for the North Atlantic region (Katona & Kraus, 1979; Katona, Harcourt, Perkins, & Kraus, 1980). Now, the Allied Whale of the College of the Atlantic curates the catalog and there are more than 8000 unique animals present in their up-to-date list (College of the Atlantic, 2018).

Running over all those photos of flukes manually to identify individual whales requires a huge effort beyond human capacity. As the number of collected images increases

with the developing technology and the significant amount of recovery in the overall whale population, addressing this need with a limited number of scientists at hand is becoming challenging day by day. Furthermore, accurate identification is controversial because the distinctive power of conventional methods is quite dependent on photo quality, evaluation of the markings in time and the ability of the observers (Friday, Smith, Stevick, & Allen, 2000).

For the past 40 years, those photos identified manually by the scientists to be tagged with an ID according to certain marks in the flukes. This requires evaluating negatives of the photos under a magnifying glass to match the patterns. To ease the burden of manual image processing, fully automated techniques are developed. One of the modern methods is matching the sequence of patterns at the trailing edges of the flukes using integral curves calculated from the outline of the edges (Weideman, et al., 2017; Jablons, 2016). Another species-unspecific algorithm called HotSpotter, matches individuals by patterns on their body parts (Crall, Stewart, Y. Berger-Wolf, Rubenstein, & Sundaresan, 2013). Batboua improved these two existing methods by stacking a Support Vector Machine (SVM) algorithm for classifying feature vectors of the images obtained from HotSpotter and curve-matching algorithms, respectively (2017).

1.2. The Humpback Whales

The humpback whale, *Megaptera novaeangliae*, is a large and thick warm-blooded mammal. Their females are larger than the males and can weigh upto 45 tons. The adult females can reach upto 19 meters while male humpback whales remain around 17.5 meters long. Their exact lifespan is unknown, but it is claimed to be in the range of 45 to 100 years (Dawes & Campbell, 2008) They are classified under baleen whales because they possess keratin palettes called baleen instead of teeth in their mouth. They live in shallow dense waters near coastal regions, close to the sea surface. They even sleep lying at the surface for only short periods. Moreover, they do not dive more than 100 meters. (Martin, 2002). Therefore, they are the widely studied whale species. They are famous for their *breaching*, jumping totally out of water and landing back on their sides creating a huge splash. They can sing not only in water but also in air, too. The function of their singing is unexplained by researchers, yet it is thought to serve for socialization (Martin, 2002). Male's songs can also be interpreted as mating signal, territorial marking or immigrational calling. These

gentle giants can take great lengths for mating and feeding. On their long journey Sun's position and the geomagnetic patterns of the Earth guide their way. They mostly feed on krill, but their diet is not restricted with only zooplanktons. They can also eat capelin, anchovy, cod, herring, mackerel, pilchards and small shrimp consuming 2000 kilograms of food per day, if available (Martin, 2002). They are curious and social animals which perform cooperative feeding. Once an individual of a pod is captured during hunting, other members have been observed to come to help or assist injured animal (Martin, 2002). The motives of their behavior is unknown. Their underwater life is still a mystery to the humankind.

The color pigmentation varies among their population. One can observe black, dark gray or brown, and even white versions of these beautiful creatures in the whole Southern Hemisphere. The back of their flukes appears in darker color such as black or dark gray, but the underside of the flukes is white (Martin S., 2002). The tail of each humpback whale is visibly unique because there are individual markings of different shapes or spots. The small round shapes on the tails are due to cookie cutter sharks, for example. Besides, most of the times barnacles sticks firmly to their flukes.

1.3. Data Source Information

Happy Whale is an online platform which collects photographs of the whales around the world and utilizes advanced image processing algorithms to contribute to the whale surveillance problem (Happywhale, 2018). Not only scientists but also individual whale watchers, naturalists or even passengers can submit photos to their website. Processing the unique marks, the pigmentation patterns, etc. found in dorsal fin or flukes with image recognition algorithm, their collaborating scientists at Cascadia Research Collective and Allied Whale identify those photos to deduce the story of whales: which ones survived during the year, what are their population trends, what are their migration, feeding or breeding patterns, etc. (Cascadian Research Collective, 2018; College of the Atlantic, 2018). Through connecting humans and the whales, this platform claims to bring attention to the marine ecosystem and also to the global challenges as we all humans face. They donated their data to a Kaggle competition and defined the problem so that many independent researchers, students, and other image recognition enthusiasts could work on it (Kaggle Inc., 2018).

1.4. Description of the Dataset

The dataset is composed of training and test images. Alternatively, an Excel file, *train.csv*, is also provided to link individual whale IDs with the training image file names. Unfortunately test images are not labeled in this Kaggle competitions. There are 9850 images of humpback whale flukes in the training set and 15610 in the test set. There are 4251 unique whale IDs in the dataset. Unluckily, each unique whale ID has only a few numbers of images associated with it and this makes the identification problem challenging. The unlabeled images which do not match with any of the IDs are tagged as *new_whale*. Some of the images are black and white, while some of them are colored. Sampling 1% of the whole dataset five times randomly, grayscale percentage is found to be 50.2%. Some random examples from the training set can be seen in Figure 1.4.1.

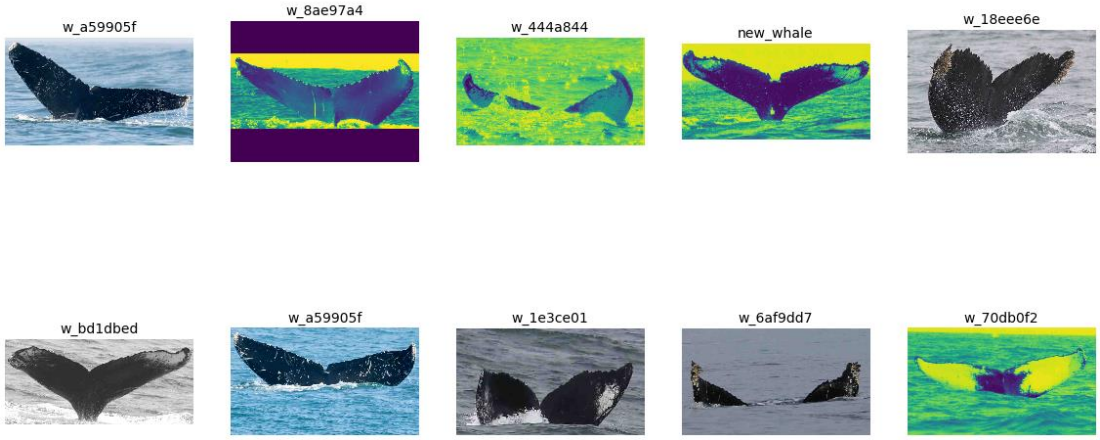

Figure 1.4.1 Random examples from the dataset

Apart from underrepresentation, the dataset is also imbalanced. Number of observations belonging to some of the classes are significantly lower than the rest. There are even classes with only one or two training images. The most frequent category, *new_whale* appears with 810 samples and *w_1287fbc* follows it with a frequency of 34.

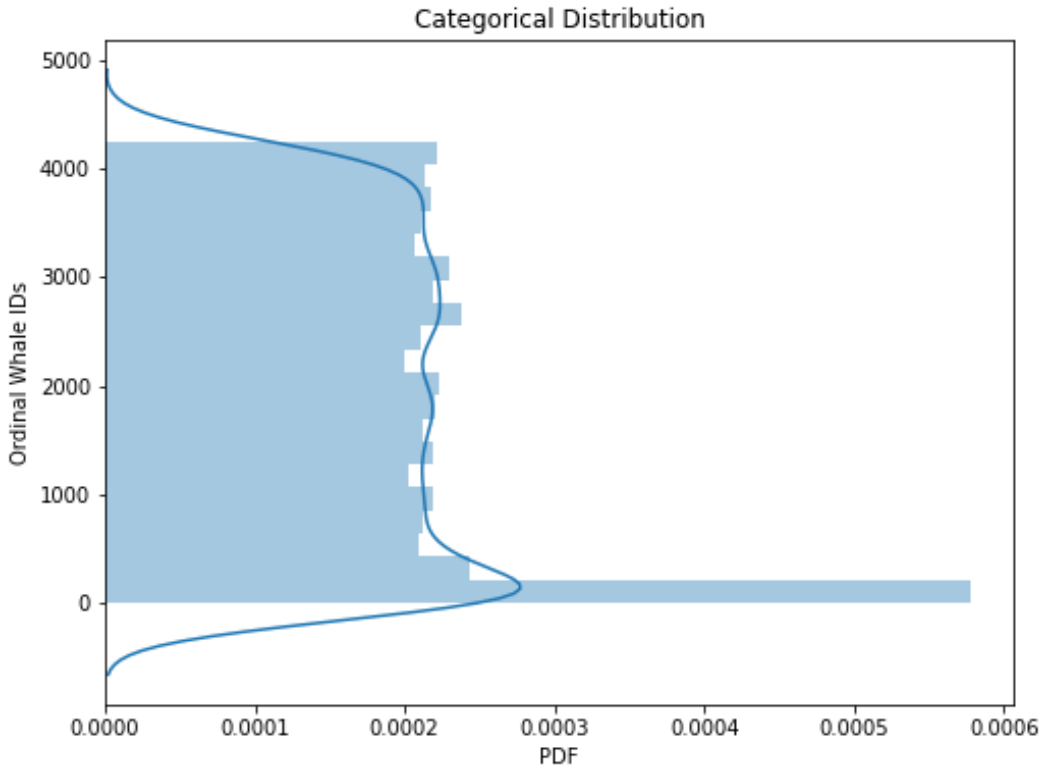

Figure 1.4.2 Categorical distribution of images' PDE (blue solid line)

Unfortunately, 2220 categories have only one image for training, 1034 categories have two, 492 categories have only three and so on… Number of categories and the number of images in them is visualized in Figure 1.4.3. Probability density estimate (PDE) of the categorical distribution is obtained by Kernel density estimate (KDE) and visualized in Fig. 1.4.2. Freedman-Diaconis rule is used for selection of the bin size in the histogram (Freedman & Diaconis, 1981). The size and the resolution values also vary from image to image. Sampling 25% of the training images randomly, the most frequent resolution value is found as 700x1050 with more than 250 instances, secondly 600x1050 follows it with almost 250 occurrences as seen in Fig. 1.4.4.

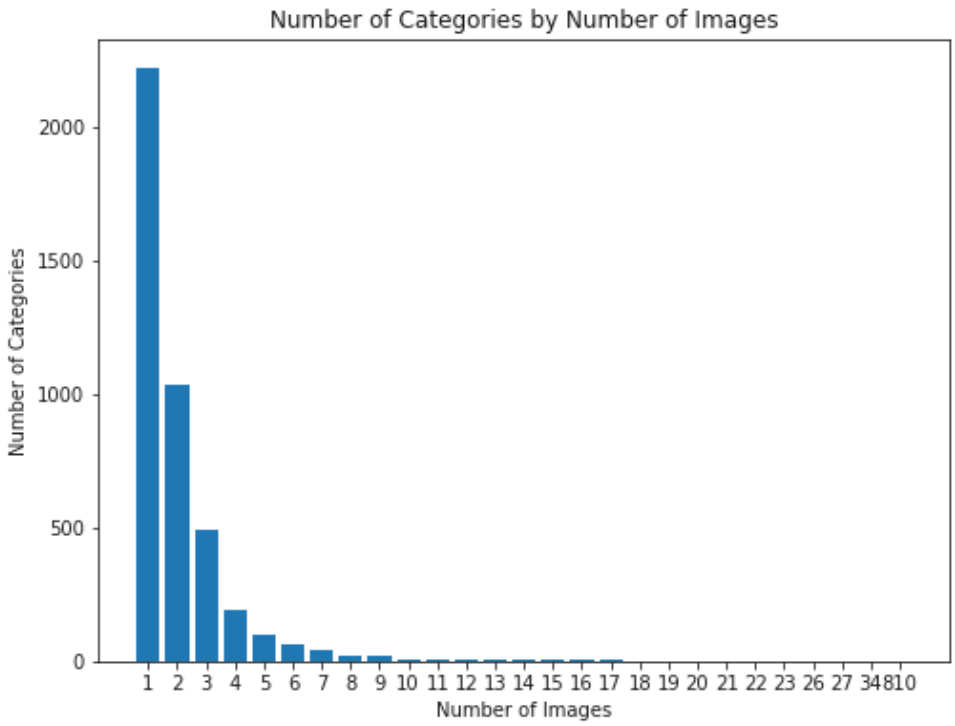

Figure 1.4.3 Count of the categories by available images in the training set

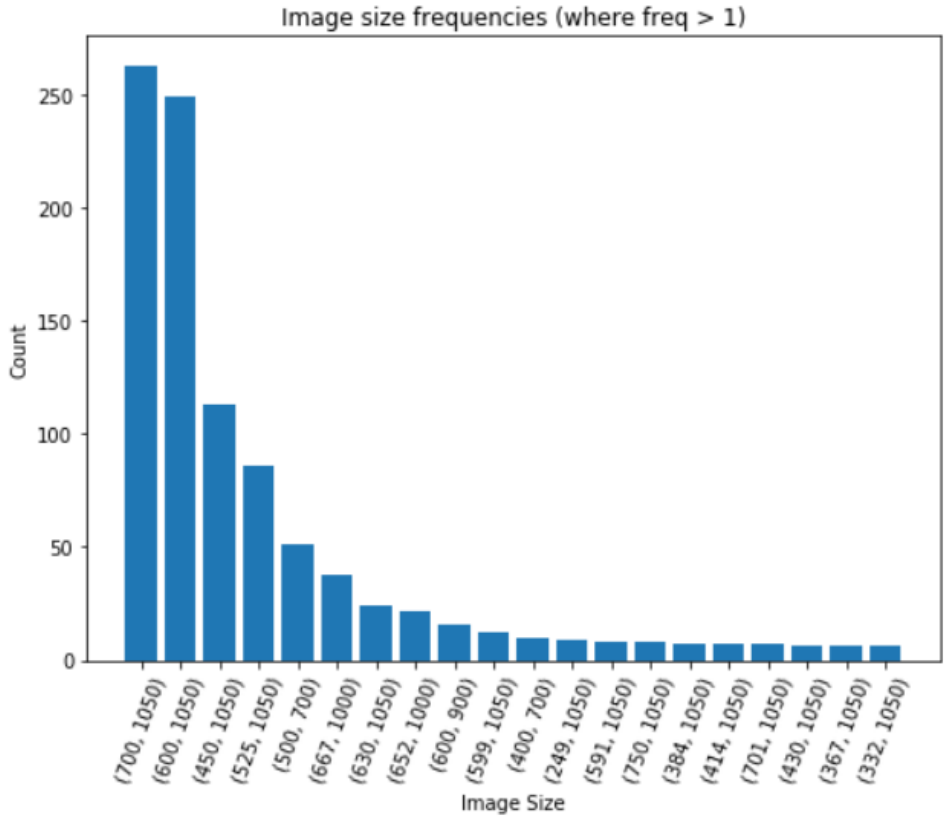

Figure 1.4.4 Occurrence frequencies of the images in the descending order

2. PROJECT DEFINITION

2.1. Problem Statement

Class imbalance is the nightmare of most machine learning algorithms. If the event to be predicted has a representation rate below 5% in the dataset, it is referred as a rare event (Mukherjee, 2018). Models trained on imbalanced datasets are known to be biased towards majority classes. As a result, the minority classes tend to be misclassified in the test set.

The Humpback Whale dataset has a serious imbalance issue. Out of 4251 whale categories 2220 IDs have only one image per class. There are also classes containing less than 5 images. Since the training set is the only labeled dataset available, we need to split it into three parts for training, validation and test. Each whale should be represented in training, validation and test sets, ideally. To resolve imbalance and representation problems altogether, this identification problem is reframed by constraining the dataset to the whale categories containing 20 or more images.

Deep Neural Networks performs well only when the number of inputs feeding their architecture is sufficiently high enough. Advanced data augmentation techniques are utilized to create synthetic images (Anonymous, 2018).

2.2. Project Objectives

The aim of this project is to help the scientists by automatizing the process of photo-identification of the humpback whales by using deep neural networks and in this way contributing to the conservation of the marine life.

2.3. Project Scope

This project only focuses on the classification of the humpback whale images. The *Region of Interest* (ROI), which is a user specified rectangle to limit the model to learn from the features within its boundaries, is whale flukes for the problem of interest. Refining calculations to ROI enhances the performance (Brinkmann, 2008). While most of the images in the dataset cropped tight around the animal flukes, in some of them the whale fluke occupies only a small portion and those images requires further cropping. Finding coordinates of ROI for each image is beyond the scope of this project. The bounding box coordinates are borrowed from Piotte's work (2018). Piotte manually cropped 1200 photos

from the dataset using a Java application and then trained a CNN model to find the bounding box coordinates for rest of the images (2018).

2.4. Project Environment

In this project PILLOW which is a friendly fork of the image processing library called Python Imaging Library (PIL) is extensively used for all processing procedures such as opening, reading, converting, transforming, cropping and finally saving the images (Clark & Contributors, 2010).

Tensorflow, which is developed by Google Brain, is an open source numerical computation library which makes machine learning faster and easier using high-performance C++ native code (Tensorflow, 2018). It is called *Tensorflow* because the numerical computation is carried out by data flow graphs whose nodes represents the operations while the edges represents the data as tensors flowing in between those nodes. However, working with Tensorflow has a steep learning curve since it requires a solid background in linear algebra and tensor calculus. Building models in Tensorflow is not easy for many of those who are new to deep learning concepts.

In this project, user-friendly, model-level deep learning library Keras is utilized. It is written in Python and operates on top of the TensorFlow backend engine with GPU support (Keras Team, 2018; Tensorflow, 2018). The modular structure of Keras and its sound documentation is advantageous for beginners of deep learning. Especially, with the help of *Sequential* model network layers can be stacked on top of each other easily.

Apart from those, we benefited from Scikit-learn library for data preprocessing and model evaluation; we have preferred to transform the images with SciPy library and we also utilized seaborn visualization library to form informative statistical graphics (Pedregosa, et al., 2011; Jones, Oliphant, Peterson, & Others, 2001).

3. METHODOLOGY

3.1. Exploratory Data Analysis on the Constrained Dataset

First *new_whale* category is dropped from the dataset since it includes all the images of the animals that are not listed in the scientist's database, and therefore it is not informative for the classification problem. Later only the categories containing 20 or more images are selected. Then we are left with 11 whale IDs and total 259 images in the dataset. The most frequent images belong to *w_1287fbc* category with 34 occurrences. The number of images in each whale category is given in Table 3.1.1, below.

Table 3.1.1 Number of images contained by whale IDs

RANK	WHALE ID	NUMBER OF IMAGES
1	w_1287fbc	34
2	w_98baff9	27
3	w_7554f44	26
4	w_1eafe46	23
5	w_693c9ee	22
6	w_ab4cae2	22
7	w_fd1cb9d	22
8	w_73d5489	21
9	w_43be268	21
10	w_987a36f	21
11	w_f19faeb	20

After constraining the dataset to 11 unique categories the severe class imbalance problem is partially resolved. Majority of the classes contain 20 – 25 photos while just one whale ID (w_1287fbc) with more than 30 images is present. The image count distribution by whale category is visualized in Fig. 3.1.1 in descending order.

When the size of the images investigated from Fig. 3.1.2, 700x1050 is found to be most frequent. Average aspect ratio of the images in the constrained dataset is found as 2.09.

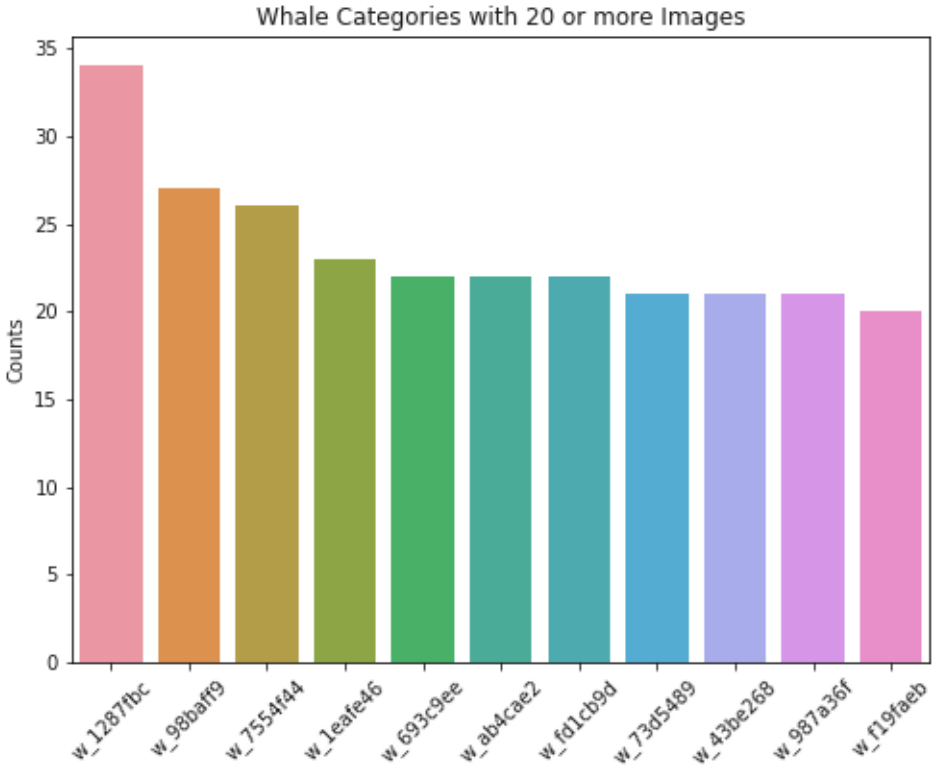

Figure 3.1.1 Image count of each whale ID

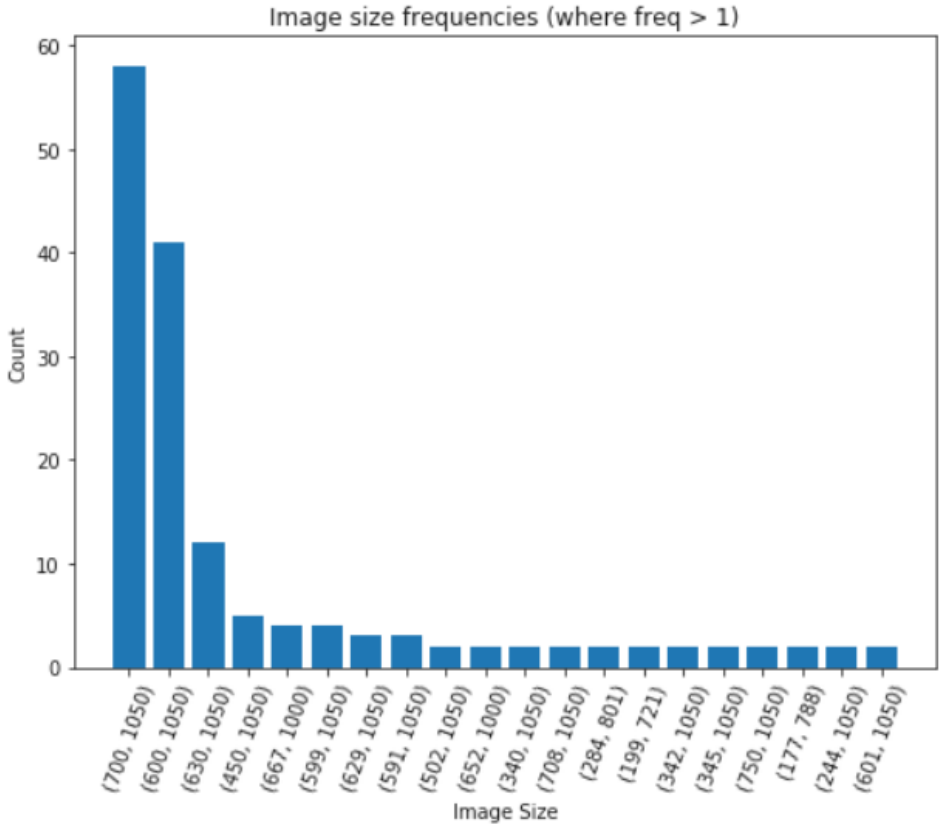

Figure 3.1.2 Occurrence rates of image sizes available in the constrained dataset

3.2. Image Pre-processing

3.2.1. Grayscale Conversion

Upon sampling 1% of the whole training images and checking their RGB codes it is found that average grayscale ratio is 52.04%. So, we can claim that almost half of the images are black and white, and the other half is colored. Since the humpback whale flukes are found mostly in black or mostly in white nature, we would not lose much information by converting three channel images into one and gain more in computation time. Using Pillow Image Library, all colored images are converted to grayscale.

3.2.2. Outlier Detection

According to photo upload instructions of the Happy Whale website, the flukes must be photographed when the whale dives into the sea and when their tail rises above the waterline (happywhale, n.d.). Besides, underside the flukes must be seen in those photos. Unfortunately, some of the images in the training set do not meet these criteria. Images of whales, broken fragments of dead whale flukes, photos taken at weird angles, photos of multiple animals, etc. are all excluded. Upon our visual inspection, 47 outliers are found among all training images and file names stored in a text file. Fortunately, there are only two such outlier images from two different classes in the constrained dataset. In order not to force the model to learn wrong features from those images, images shown in Fig. 3.2.1 are excluded from the dataset.

Figure 3.2.1 Outlier images in the constrained dataset

3.2.3. Rotation

Note that in some of the images edges of the flukes points down. Upon visual inspection they are recorded and to increase the accuracy of classification, rotated by 180 degrees. Those upside-down images can be seen in Fig. 3.2.2.

Figure 3.2.2 Upside down images found in the constrained dataset

An image reading function is written to automatically flip the images that are found in the rotation list as seen in Fig. 3.2.3.

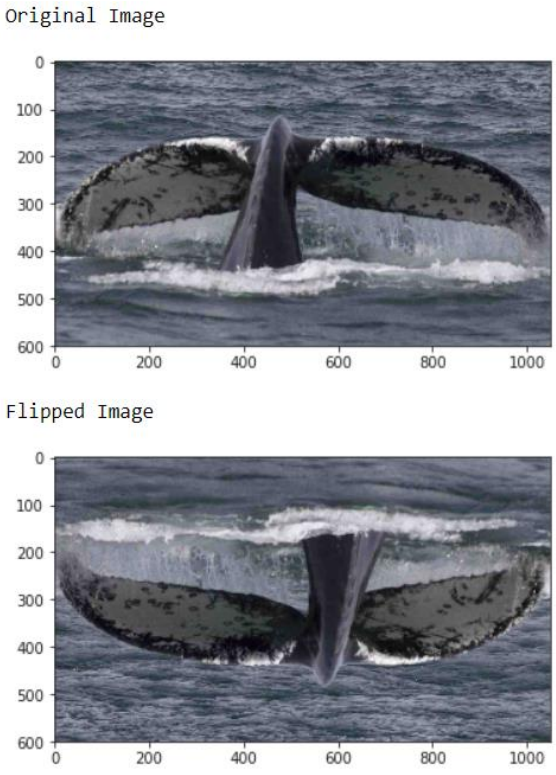

Figure 3.2.3 Automatic rotation of required image files while reading

3.2.4. Image Cropping and Reshaping

Images are cropped to ROI, i.e. the whale flukes, using bounding-box coordinates taken from a previous work (Piotte, Bounding box data for the whale flukes, 2018). A margin

is left around the rectangle to save us from cropping the edges of the flukes accidentally. Otherwise, information loss due to faulty cropping gives more harm than the gain due to tight cropping. That is why a margin of error is defined around each image to compensate for bounding box errors. Upon experimenting with the borders, 10% of the height for the top and bottom borders, 10% of the width for the left border and 15% of the width again for the right border is chosen to be optimum values that minimizes the cropping errors. Despite careful inspection and various trials with compression rate and margin ratios some of the images are failed to be cropped properly. Those are listed in a separate file and dropped from the dataset to prevent the model from learning misguiding features. After omitting the erroneously cropped images, we are left with 250 samples in total. Class distributions do not change much after all those extractions. We still have more than 20 images in each class as seen in Fig. 3.2.4.

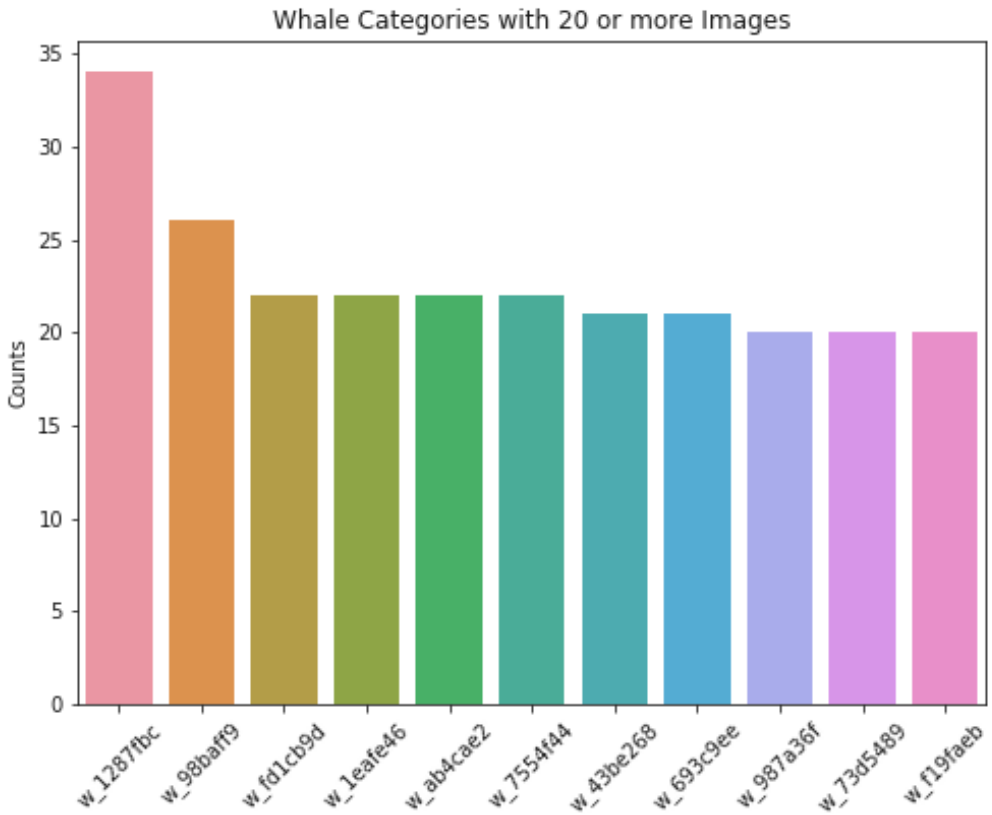

Figure 3.2.4 Final categorical count of the dataset

Using affine transformation, rectangular images are further mapped to a square with 224x224 resolution (considering single channel for black & white). The size of the images is adjusted to match the input size of the model to be used (i.e. ResNET). The average width

over height aspect ratio of the images in the dataset was calculated to be 2.09 previously. This value is used as a horizontal compression ratio while the square images are created.

3.2.5. Affine Transformation

Prior to cropping to the bounding box, coordinates of the bottom left corner (x_0, y_0) and coordinates of the upper right corner (x_1, y_1) are obtained from Piotte's work (2018). These coordinates are enlarged to the margins as mentioned above and then affine transformation is performed. Affinity, meaning likeness or similarity is a mapping between affine spaces which preserves collinearity and the ratios in between distances of points, straight lines and planes. Affine transformation can be described mathematically as given as (Weisstein, n.d.),

> "If X and Y are affine spaces, then every affine transformation $f: X \to X'$ is of the form $x \to Tx + b$, where T is a linear transformation on the space X, x is a vector in X, and b is a vector in X'."

The transformation matrix can be represented in an augmented form such that:

$$\begin{bmatrix} \boldsymbol{x}' \\ 1 \end{bmatrix} = \begin{bmatrix} T & \boldsymbol{b} \\ 0 \cdots 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ 1 \end{bmatrix}$$

The equation above is equivalent to the following linear equation, expressed in a compact form,

$$\boldsymbol{x}' = T\boldsymbol{x} + b$$

With the help of the augmented matrices, both the translation and the linear mapping can be represented as using a single matrix multiplication. For the problem of interest, the transformation matrix is given below.

$$\mathbf{T} := \begin{bmatrix} 1 & 0 & {(y_1 + y_0)}/{2} \\ 0 & 1 & {(x_1 + x_0)}/{2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} {(y_1 - y_0)}/{224} & 0 & 0 \\ 0 & {(x_1 - x_0)}/{224} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & {-224}/{2} \\ 0 & 1 & {-224}/{2} \\ 0 & 0 & 1 \end{bmatrix}$$

Here, the first and last matrices are for translation whereas the matrix in the middle scales the ROI to the output image size (224x224). Applying the operator above to all of the images, we obtain cropped and squared versions of them with 224x224 resolution.

3.2.6. Standardization

As a final step, to suppress different illumination effects in each image, mean value is subtracted from each pixel value and the result further divided by the variance, as follows,

$$z = \frac{(x - \mu)}{\sigma}$$

Here x is the initial pixel value, μ is the mean value of the image, σ is the standard deviation and z is called the standardized pixel value. In this way images are standardized to zero mean and unit variance. Since the image matrices do not possess a sparse characteristic, custom standardization is preferred over plain normalization and scaling.

3.3. Image Augmentation

To make the model more robust, the dataset is expanded with *ImageDataGenerator* class of Keras. Some noise is added, images are distorted by random shifting, rotating and flipping. These transformations are done because change in the perspective can change the apparent shape of the flukes. Here images are allowed to rotate 20 degrees, randomly zoomed up to 20%, shifted up to 20% of their height and width and flipped horizontally. These random transformations are skipped during the testing phase.

3.4. Artificial Neural Networks

The heart of the deep learning lies in artificial neural networks (ANN) since they are the starting point of all. Their architecture is inspired from biological neural networks. In our brains, learning happens in response to external stimuli. Below a simplified diagram shows the structure of a neuron.

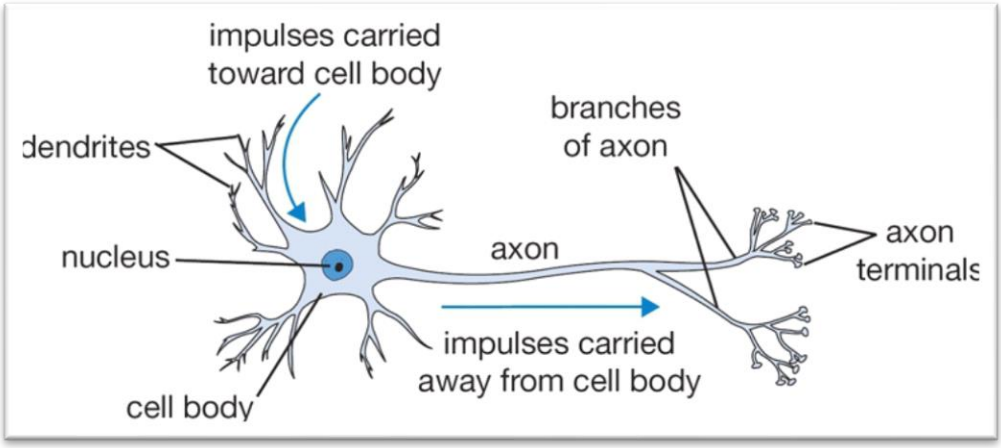

Figure 3.4.1 Biological neuron

Neurons are found in the cerebral cortex of the brain. Differing from other cells, neurons possess very long axon to carry signals. At the very end of each axon branches called telodendria lies. At the tips of each telodendrion, there are synaptic terminals which enable neurons to talk with other neurons. When a neuron receives sufficient amount of signals from others within a short amount of time via its dendrites, it fires its own signal for the adjacent cell.

Each neuron is connected to thousands of other neurons to form vast networks. The size of such networks reaches to billions of units, so that the brain can perform complex tasks. Below vertical cross-section of Golgi-stained cortex of an infant is shown (reproduced from (Cajal, 1899)). Here, the neurons form a laminar structure and each layer is densely connected to one another via synaptic terminals.

Figure 3.4.2 Multilayer structure of human cerebral cortex

In 1943, McCulloch and Pitts first introduced ANNs mathematically with a model called propositional logic (McCulloch & Pitts, 1943). In their pioneering paper, the authors suggested a simple computational model to explain how neurons cooperate with each other to perform complex computations (McCulloch & Pitts, 1943).

In 1957, Rosenblatt invented the perceptron, a single layer of linear threshold (LTU) units (the artificial neuron shown in Fig. 3.4.3) with each neuron is connected to all inputs. The LTU collects all the signals coming from all inputs as the biological neuron does and takes a weighted sum of them. If the resulting sum exceeds a threshold called bias the unit is activated and outputs 1, otherwise it just outputs 0. In other words, LTU behaves like a classifier and bias is a measure of how easy it is to get the artificial unit to fire. With different combinations of LTUs, logical operations such as AND, OR, XOR, etc. can be performed. Later, the research on the subject is escalated starting from1960s and multilayer perceptrons

are developed as shown in Fig. 3.4.4 (Geron, 2017). Notice that the units of each layer are densely connected to the units of neighboring layers. These architectures are called ANNs.

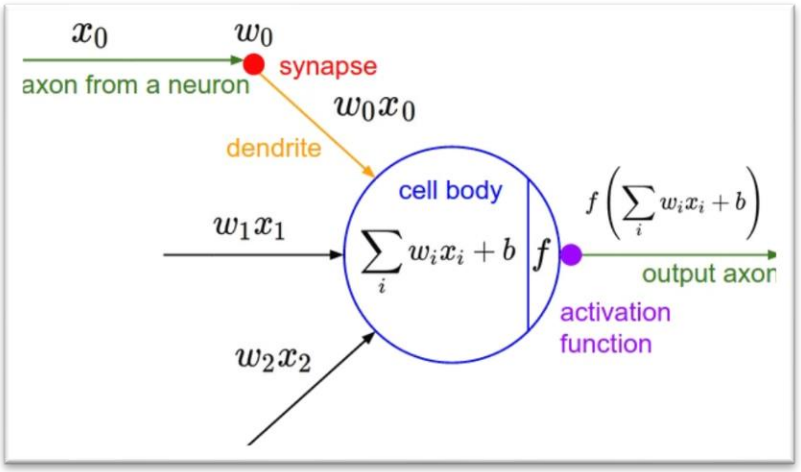

Figure 3.4.3 Linear threshold unit

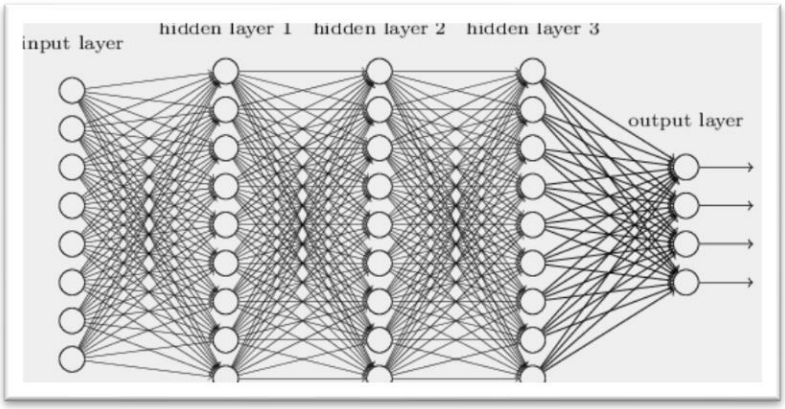

Figure 3.4.4 ANN with three hidden layers

3.5. Convolutional Neural Networks

Studies on brain's visual cortex led to another classes of deep, feed-forward artificial neural networks called convolutional neural networks (CNN). Differing from ANNs which have fully connected layers, CNNs are able to take into account the spatial structure of the input data. Thus, application of CNN architectures to visual imagery is shown to be pretty successful up to now.

During 1958 and 1959 Hubel and Wiesel made series of curial and yet crucial experiments on cats and later in 1968 on monkeys to study the underlying structure of virtual cortex (Hubel, 1959; Hubel & Wiesel, 1959; Wiesel & Hubel, 1968). They presented different visual images to an anesthetized cat with a deep brain probe and observed that individual nerve cells fire vigorously only to the lines at particular orientation within their receptive field (Blakemore, 1973). Their findings can further be summarized as follows:

i. Neurons organized in a columnar architecture, act together to pursue a perception.
ii. Many neurons possess a small receptive field.
iii. Some neurons react to only specific line orientations
iv. Other neurons with larger receptive fields might react to more complex patterns.

The biological idea behind the CNN architecture is the feature specific neurons and the notion of the receptive field as discussed above. Network layers are stacking over each other inspired by the columnar structure of the biological neurons in the visual cortex. Each unit in the first hidden layer of a CNN will be connected to a small region of the inputs (receptive field) as shown in the figure below. A receptive field is rectangular area sliding across the input image with defined stride steps.

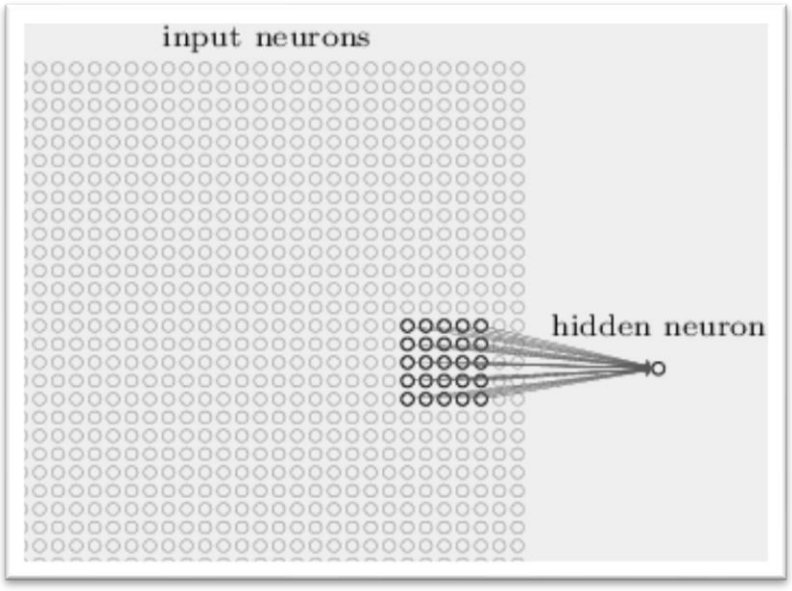

Figure 3.5.1 Receptive field

Learning is done by the associated weight of each connection. Besides, hidden layer also learns a bias. Through the filter weighted sums are calculated, and feature map is created (the convolutional part of the network maps the features). Deep down the network, those

filters become larger (i.e. with large receptive fields), so they can consider signal coming from a larger area and reproduce more complex features. Finally, the networks outputs N-dimensional vector for N classes and the densely connected part at the very end of each CNN detects those high-level features.

A typical CNN architecture is constructed by stacking convolutional layers on top of each other. Output of each layer is the input of the next. The input image is passed to the first layer by *input layer*. This stacked architecture of convolutional layers makes it easy for CNN to focus on local features in the first hidden layer, then assemble them to learn more complex features in the next hidden layer. Global features of the image are deduced via this combined approach.

As we go deep down in the network, the size of the layers shrinks (due to convolution) and network starts to ignore features at the edges (pixels with no neighbors). To avoid these *padding* is used. Simply a buffer layer of zeros is put around the input image, so that the size of the output layer will be the same with the input layer.

Shared filter weights are initialized in each convolutional layer. For example, ResNET uses MSRA initialization. It is a Gaussian type initialization which keeps variances through transformation (Li, Jiao, Han, & Weissman, 2016).

After each convolutional layer, an activation layer is applied. For ResNET, rectified linear unit (ReLu) is used for the sake of computational efficiency (except SoftMax is used in the last layer). ReLu outperforms other conventional non-linear activation functions such as *sigmoid* and *tanh* (Xavier, Bordes, & Bengio, 2011). ReLu also said to be the solution of the *vanishing gradient* problem which stops the neural network from further training. In this layer $f(x) = max(0, x)$ is applied to all of the values in the input. All negative activations are replaced by zero. So, nonlinearity is introduced to the system without affecting the receptive fields of the convolutional layer.

Pooling layer is another key element of CNNs. Output of the previous layer is down-sampled to reduce computational load and number of parameters. By the virtue of pooling, the risk of over-fitting is reduced somehow. Shrinking the input image also introduces a level of location invariance, so that the neural network can tolerate a little bit of image shift (Geron, 2017). In ResNET *average pooling* is used as aggregation function. Just like in convolutional layer, a rectangular window with defined size, padding and stride is slid across the input layer, and average input value in each kernel makes it to the next layer.

When a complex model fits the noise in the data instead of the underlying relationship of the features, overfitting occurs. This can be avoided by a regularization technique such as *batch normalization* as ResNET does. Using an initializer with ReLU reduces the vanishing/exploding gradient problem at the beginning of the training but not during the training. The input distribution of each layer changes during training as the parameters of the previous layer changes leading to a drop in the learning rate. This problem is addressed as *Internal Covariate Shift* in the 2015 paper of Ioffe and Szegedy within which normalization for each training mini-batch is suggested (Ioffe & Szegedy, 2015). The authors claim that this method not only is a solution to vanishing/exploding gradient problem but also acts somehow as a regularization technique that eliminates the need for dropout. Upon experiment, the same effect is observed and dropout layers after convolutional layers are excluded. Before activation function of each layer inputs are zero-centered, normalized, scaled and shifted by batch-normalization scaling and shifting parameters per layer. The required means and standard deviations of the inputs for zero-centering and normalization steps are estimated from the current mini-batch.

Finally, the output of the last pooling layer is flattened to feed the fully connected layers at the end. High level features coming from the activation maps of the previous layer are detected. In the final *dense* layer, the net output is given as the probability distribution of each class. Using SoftMax approach the class with highest probability is chosen. That is how a CNN transforms an input image into a vector of features describing the whale.

3.6. ResNET50

Residual Network (ResNET) is an enhanced 50 layered CNN trained on ImageNET dataset. It solves the degradation problem of CNNs using shortcuts between layers and shortens the training time via bottleneck design. Degradation problem is defined by the authors of the original paper of ResNET as (He, Zang, Ren, & Sun, 2016):

> "When deeper networks are able to start converging, a degradation problem has been exposed: with the network depth increasing, accuracy gets saturated (which might be unsurprising) and then degrades rapidly."

In a typical network, weights are updated proportional to the partial derivative of the error function with respect to the error function after each epoch. However, if the cost function that is to be optimized during training has a plateau, then these gradients become very small, hence weights will not be updated effectively. This is the vanishing gradient problem. Specifically learning decelerates through the layers of the deep neural network due to very slow gradient descent. ResNET both solves those saturation and degradation problems via residual learning.

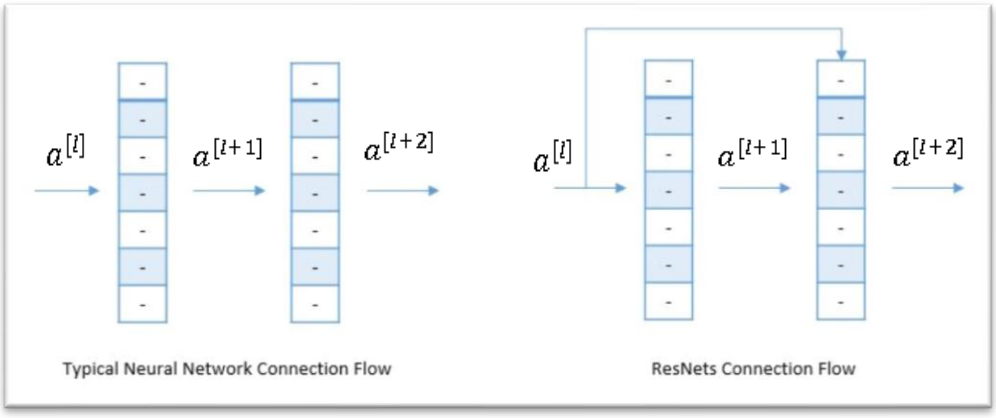

Figure 3.6.1 Plain network vs. ResNET

As Andrew NG beautifully put it in his online course on deep learning, layers of plain networks transfer their input from one another as activation in an order, however ResNET is composed of residual blocks stacked over each other (NG, 2017). As shown in Fig. 3.4.3, LTUs of a layer linearly transforms the incoming activation at first then the nonlinearity is applied (i.e. ReLU). After passing a layer activation a^l becomes a^{l+1} in two step process:

$$z^{[l+1]} = W^{[l+1]}a^{[l]} + b^{[l]} \quad \text{and} \quad a^{[l+1]} = f(z^{[l+1]})$$

Here W is the weight matrix, b is the bias term and f is the nonlinear activation function (i.e. ReLu). Similarly, final activation can be formulated as,

$$z^{[l+2]} = W^{[l+2]}a^{[l+1]} + b^{[l+1]} \qquad \text{and} \qquad a^{[l+2]} = f(z^{[l+2]})$$

In ResNET, there is a shortcut is added before the nonlinearity as shown in Fig. 3.6.1, so the final activation becomes,

$$a^{[l+2]} = f(z^{[l+2]} + a^{[l]})$$

Even if $z^{[l+2]}$ vanishes due to L_2 regularization, addition of the residual term, $a^{[l]}$, at the end guaranties that the network does not degrade as plain networks does since

$$a^{[l+2]} = f\left(a^{[l]}\right) = a^{[l]}$$

The equation above implies that the residual network learns the identity function well and this brings a lower bound on its learning process. That is the fundamental concept of ResNET.

In this study, we loaded ResNET50 model from Keras repository. The input size was set to 224x224x1. The implementation can be found in Appendix A and the model architecture is given in Appendix B.

4. RESULTS AND DISCUSSION

4.1. Establishing a Baseline Model

In the first part of this study the effect of different splitting schemes on the model's performance is investigated. Since a limited number (i.e.250) images are present, and performance of a deep learning model is highly dependent on the size of the input images available, the train/test split scheme was carried out with extreme care. We experimented on three different splitting plans. All splits were carried out in a stratified fashion so that 11 classes were represented with the similar distribution in the resulting sets as seen in the Fig. 3.7.1. With stratified splitting we ensured that training set is a good representation of validation and test sets.

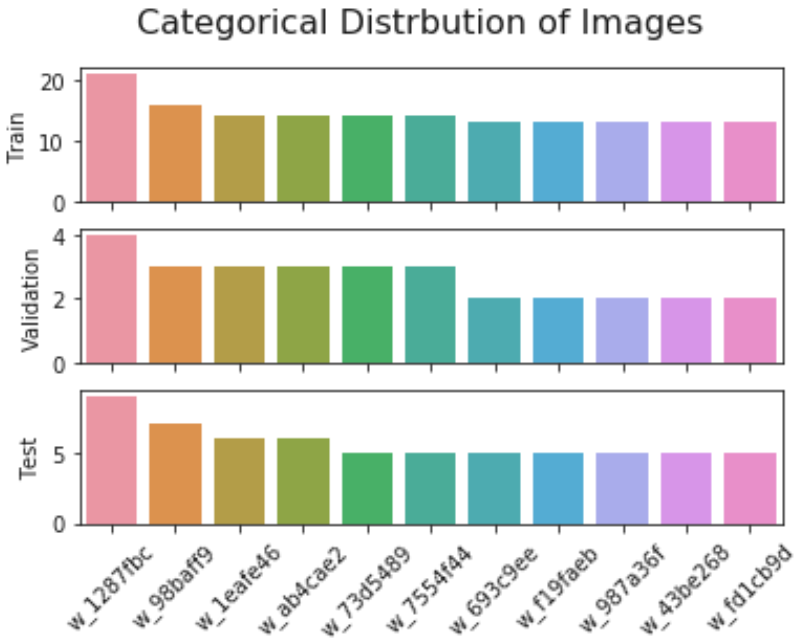

Figure 4.1.1 Distribution of the images in training, validation and test sets

Before training the model, the learning process was configured. ADAM optimizer, which is a combination of Gradient Descent with momentum and RMSProp have been chosen because of its effectiveness. Authors of ADAM observed that exponential averages of past gradients and variance of the gradients (first and second momentum) decay towards zero, so they come up with a bias correction for these terms (Kingma & Ba, 2014). However, we defined a custom decaying function decreasing the learning rate by 10% at each epoch.

Hence, learning rate decreased as it gets closer to the optimal solution. The form of the annealing function is given below.

$$\eta_{i+1} = \left(\eta^i - \eta^i \frac{10}{100}\right) = \eta_0 \left(\frac{90}{100}\right)^i$$

Here η is the learning rate, i is the epoch index (time step) and η_0 is the initial learning rate chosen to be 0.001. *LearningRateSchedular* of Keras was utilized to decay the learning rate by 10% at each epoch.

As a cost function categorical cross entropy has been chosen since it penalizes erroneous predictions more, producing larger gradients and thus converging faster. The cross entropy between the estimated probability and the target probability was minimized. Number of epochs was chosen as a large number (i.e. 300) to guarantee convergence. Unfortunately, the batch size has been kept very small due to memory limits of the available GPU (GeForce GTX 950M by NVIDIA). Therefore, the variance was increased, and we obtained spiky graphs while monitoring the model's performance during training from epoch history as seen in Fig. 4.1.2.

In the first model (RESNET_VAL), 25% of 250 images was reserved for testing, 11.25% was kept for validation to monitor the performance history during training and 63.75% was used in training. The final validation accuracy was quite poor (0.448), and the accuracy of the validation set got stuck at 0.43 after 50 epochs during training. We believe that the reason of the observed bottleneck is due to limited size of the input space, we decided to enlarge the training set. Then, the validation set was merged to training set and another model, RESNET_75_25 has been trained. The performance was improved obtaining a validation accuracy of 0.635 but this still was not as expected. Finally, decreasing the test set size to 10% of the dataset, we ran a final model called RESNET_90_10. The saturation in the validation accuracy during training still persisted somehow but final validation accuracy quickly escalated to 0.84. The performance metrics of the models deployed can be found in Table 4.1.1.

Table 4.1.1 Performance comparison of baseline models

MODEL	TRAIN/VAL./TEST RATIO (%)	TRAIN ACC.	VAL. ACC.	TRAIN LOSS	VAL. LOSS
RESNET_VAL	63.75/11.25/25	0.627	0.448	1.067	1.717
RESNET_75_25	75/25	0.840	0.635	0.496	1.107
RESNET_90_10	90/10	0.964	0.840	0.119	0.501

The confusion matrix of the best model, RESNET_90_10 can be seen in Fig. 4. When we observed the confusion matrix, we saw that erroneously classified categories, especially 6^{th} (w_7554f44) and 10^{th} (w_43be268) whale IDs, have only two test images available. Wrong labeling just one image of those categories corresponds to incorrectly predicting half of the labels. This situation substantially increased the multinomial loss (0.501).

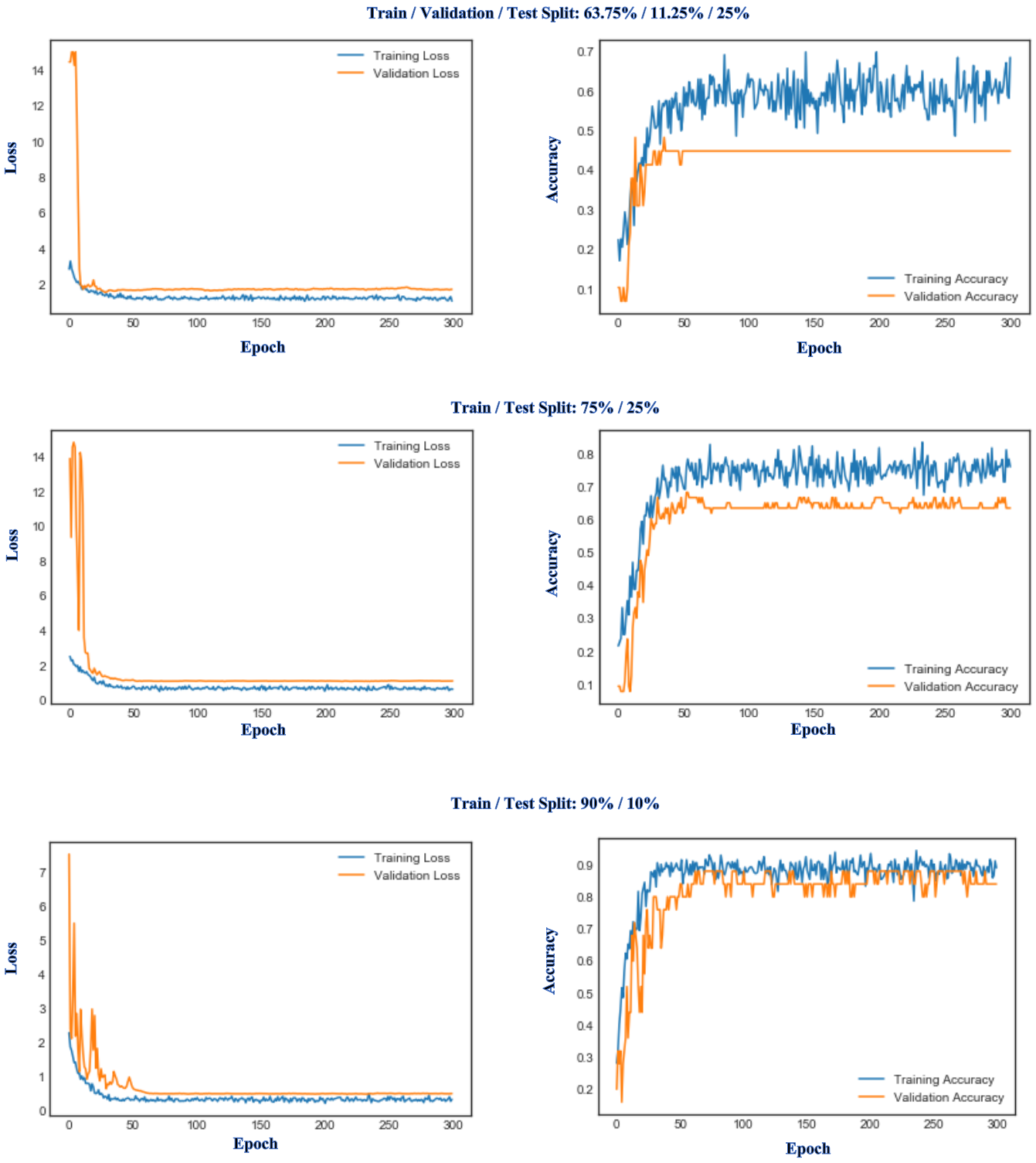

Figure 4.1.2 Performance comparison of three different splitting schemes

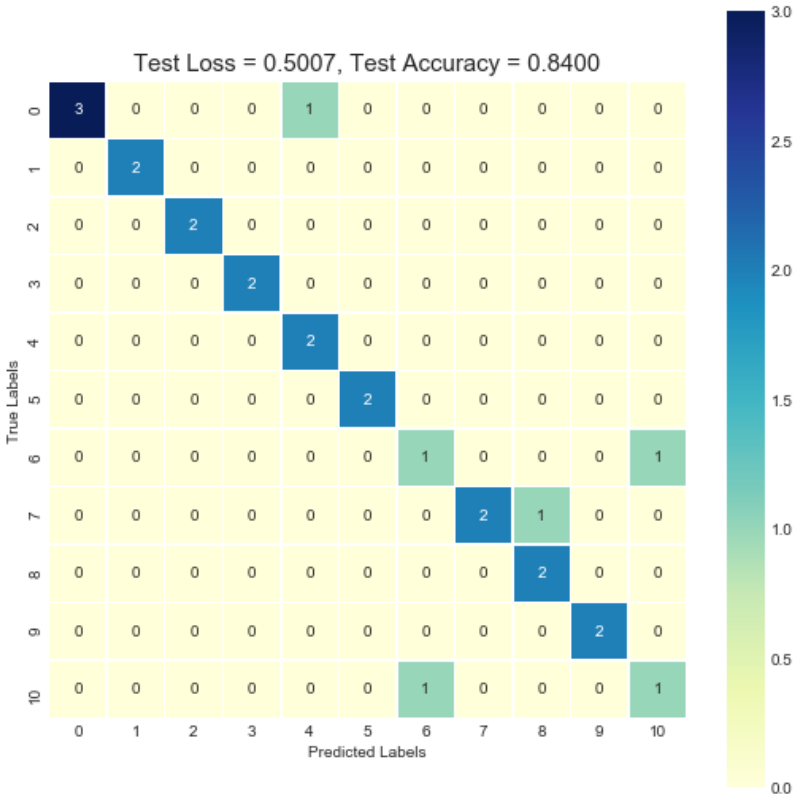

Figure 4.1.3 Confusion matrix of RESNET_90_10

Considering threshold in between training and test set size as discussed above, we decided to continue reserving 20% of the dataset for validation. This split ratio was enough to reserve at least four images for each category in the test set as seen in Fig. 4.1.4.

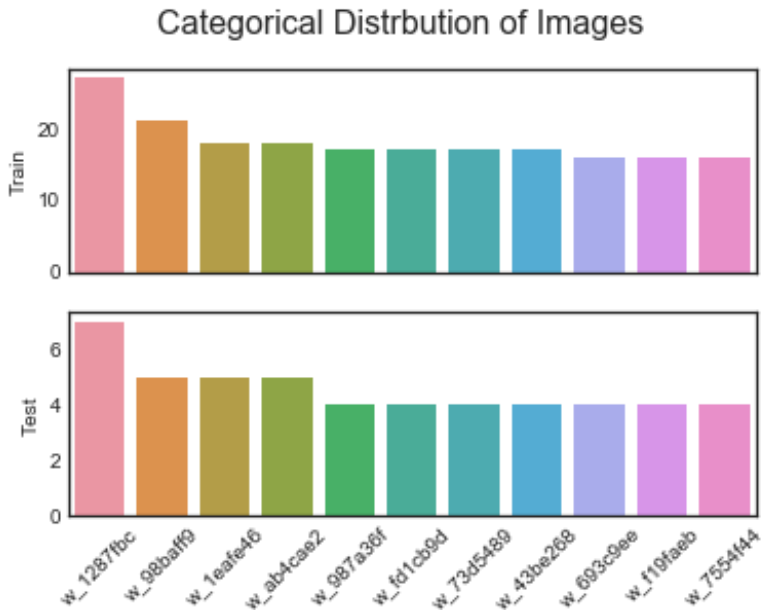

Figure 4.1.4 Image count distribution for 80/20 train/test split

4.2. Different Initializations

In the second part of the study, we trained the models once with different learning rates to ensure convergence. Then, resetting the learning rate to ADAM's default learning rate of 0.001, we reduced it by 10% every epoch during training. By doing so, we changed the position of the initial point of optimization prior to the training process. In other words, to accelerate the learning procedure we experimented on the effect of the initialization on the minimization of the cost function (i.e. convergence). We experimented by setting the initial learning rate to 0.001, to 0.005 and then to 0.01, respectively and obtained the best results for 0.005. The performance metrics are given in Table 4.2.1. for comparison purposes.

Transferring the calculations on a CPU with 64 GB of RAM we could finally increase the batch size to 50. At the same time the epoch number was decreased to 70 since we observed that previous models converged before 75 epochs.

Table 4.2.1 Performance metrics for different learning rates

MODEL	INITIAL LEARNING RATE	TRAIN ACC.	VAL. ACC.	TRAIN LOSS	VAL. LOSS
RESNET_ADAM_1E3	0.001	0.975	0.920	0.071	0.421
RESNET_ADAM_5E3	0.005	1.000	0.940	0.033	0.318
RESNET_ADAM_1E2	0.01	0.980	0.880	0.059	0.335
RESNET_ADAM_OPT	0.00432	1.000	0.940	0.024	0.341950

The model RESNET_ADAM_5E3 has been built with an initial learning rate of 0.005 outperformed all others. It reached to an accuracy value of 0.940 in the validation set while minimizing the loss to 0.318. On the other hand, the curious case of the training accuracy hitting to the unity might be due to overfitting. Unfortunately, the model's performance could not be improved more due to limited number of the training images (i.e. 200 photos). When we inspected Fig. 4.2.2. we concluded that the one and only train/test gap that is in the closing trend is that of RESNET_ADAM_1E3. Furthermore, the unstable behavior in the train/test validation and loss curves was not observed in that model since its

starting learning rate is the same with the one that is used in the annealing function, so the learning rate of the first and second epochs was not changing abruptly.

Assuming the accuracy is a quadratic function of the initial learning rate, we fitted a second order polynomial to the accuracy versus learning rate plot and calculated the value that maximizes the fit function (see Fig. 4.2.2). The climax point has a learning rate of 0.00432. Then the model, RESNET_ADAM_OPT, is trained again with this rate, but neither the accuracy nor the loss has been improved significantly when compared to the scores obtained by RESNET_ADAM_1E5. The performance metrics are given in Table 4.1.1. The closeness of the performance scores of those two models is not surprising since their initial points are nearly the same and for this reason they converge to the same solution with the same methods.

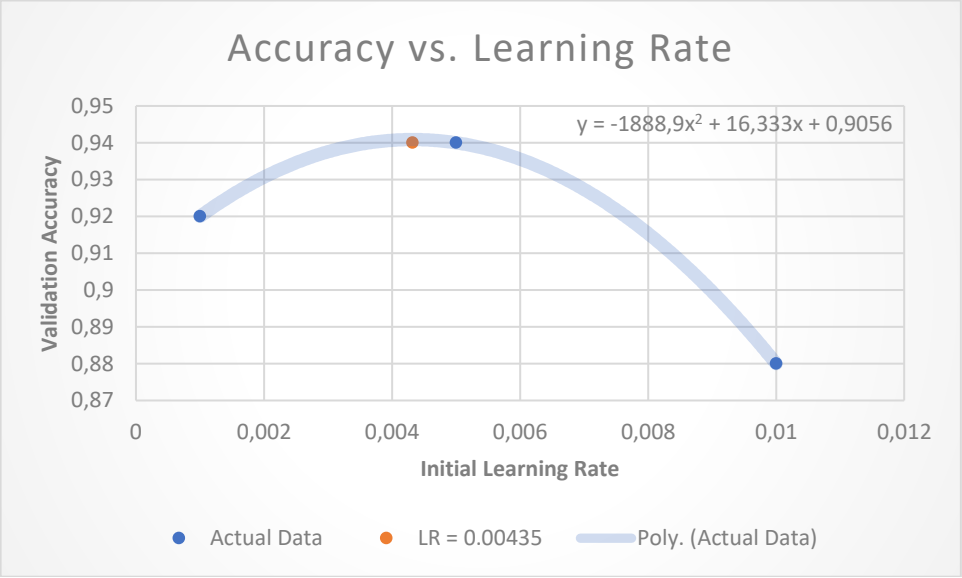

Figure 4.2.1 Change in accuracy with different initializations

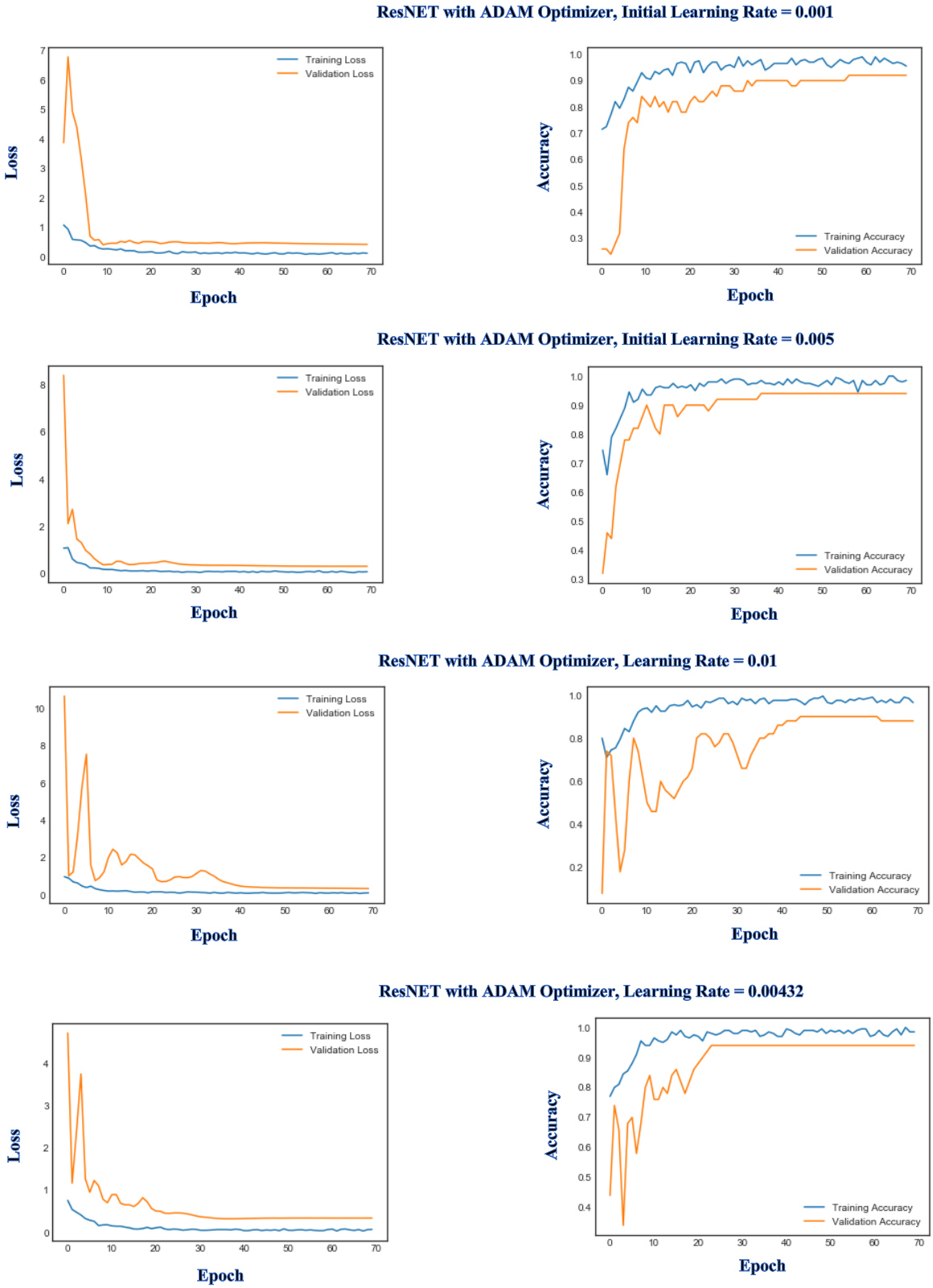

Figure 4.2.2 Performance comparison of models with different initializations

5. CONCLUSION

In conclusion, we report that ResNET_ADAM_1E5 is a successful CNN model to identify humpback whale images with an accuracy of 0.94 although it is trained with a very small dataset of 200 images, only. This, model could be enhanced further if only more images were available. Recently, Kaggle has restarted the Humpback Whale Identification Competition with an expanded dataset (Kaggle Inc., 2018). As a future work, this enlarged dataset will be used to improve the model, even making it to identify all of the whale IDs, even the ones not cataloged yet. To enhance the image preprocessing identical photos known to be present in the train set will be determined using image hashing and will be dropped. Furthermore, we will be training our own object detector such as YOLO, Faster R-CNN, Mask R-CNN or a single shot detector (SSD) to determine the bounding box coordinates.

6. SOCIAL AND ETHICAL ASPECTS

This study does not violate privacy or liberties of any individual of the humpback whale community. Since the photographs of the dataset were taken at a distance no animals were harmed. In this cruelty free study, the end product was not tested on live animals.

7. VALUE DELIVERED

By classifying the humpback whales, we contribute to Happy Whale's efforts to understand these endangered animals. To ensure their survival for future generations, their migration patterns needed to be tracked. Moreover, tracking is important to understand their pod dynamics. Photo identification is the only advanced and automatized method to help the scientist to track the whales individually and we obtained a high accuracy of 0.92 with these techniques. We are planning to pursue our studies on the enlarged dataset to aid whale conservation efforts more. Moreover, this project reminded us that all the species of this planet have equal rights to live, breed and prosper. The Earth does not belong to the humankind, only.

Bibliography

Anonymous. (2018, 5 26). *Deep learning unbalanced training data? Solve it like this.* Retrieved 12 4, 2018, from mc.ai: Aggregated news around AI and co.: https://mc.ai/deep-learning-unbalanced-training-datasolve-it-like-this/

Anonymous. (2018). *Whale Facts: Marine Mammal Facts & Information*. Retrieved 12 27, 2018, from Humpback Whale Facts: https://www.whalefacts.org/humpback-whale-facts/

Batbouta, A. (2017). Computer Assisted Labeling of Humpback Whales and Whale Sharks. Troy, New York: Rensselaer Polytechnic Institute.

Blakemore, C. (1973). Colin Blakemore does terrible things to kittehz. For science! Retrieved 12 23, 2018, from https://www.youtube.com/watch?v=QzkMo45pcUo

Brinkmann, R. (2008). *The Art and Science of Digital Compositing: Techniques for visual effects, animation and motion graphics.* Morgan Kaufmann.

Cajal, S. R. (1899). *Wikipedia*. Retrieved 12 22, 2018, from Cerabral Cortex: https://en.wikipedia.org/wiki/Cerebral_cortex

Cascadian Research Collective. (2018, July 22). Retrieved from Cascadia Research: http://www.cascadiaresearch.org/

Clark, A., & Contributors. (2010). *PILLOW.* Retrieved from GitHub: https://github.com/python-pillow/Pillow

College of the Atlantic. (2018, July 22). *Allied Whale*. Retrieved from College of the Atlantic: https://www.coa.edu/allied-whale/

Crall, J., Stewart, C., Y. Berger-Wolf, T., Rubenstein, D., & Sundaresan, S. (2013). HotSpotter - Patterned Species Instance Recognition. *IEEE Workshop on Applications of Computer Vision (WACV)* (pp. 230-237). IEEE.

Dawes, J., & Campbell, A. (2008). *Exploring the World of Aquatic Life.* New York: Chelsea House Publications.

Freedman, D., & Diaconis, P. (1981). On the Histogram as a Density Estimator: L2 Theory. *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete*, *57*(4), 453-476.

Friday, N., Smith, T. D., Stevick, P. T., & Allen, J. (2000). Measurement of photographic quality and individual distinctiveness for the photographic identification of

humpback whales, Megaptera novaeangliae. *Marine Mammal Science, 16*(2), 355-374.

Fuhr, N., Quaresma, P., Gonçalves, T., Larsen, B., Balog, K., & Macdonald, C. (2016). LifeCLEF 2016: Multimedia Life Species Identification Challenges. In L. Cappellato, & N. Ferro (Ed.), *International Conference of the Cross-Language Evaluation Forum for European Languages* (pp. 286-310). Cambridge: Springer.

Geron, A. (2017). *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems.* Sebastopol, California: O'Reilly Media, Inc.

Happywhale. (2018, July 21). *Happywhale*. Retrieved from About Happywhale: https://happywhale.com

happywhale. (n.d.). *Image Submission Guidelines*. Retrieved December 4, 2018, from Happywhale: https://happywhale.com/instructions

He, K., Zang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. *IEEE conference on computer vision and pattern recognition*, (pp. 770-778).

Hubel, D. H. (1959). Single unit activity in striate cortex of unrestrained cats. *The Journal of physiology, 2*(147), 226-238.

Hubel, D. H., & Wiesel, T. N. (1959). Receptive fields of single neurons in the cat's striate cortex. *The Journal of physiology, 3*(148), 574-591.

International Whaling Commission. (2018). *Commercial Whaling*. Retrieved 12 27, 2018, from International Whaling Commission: https://iwc.int/commercial

Ioffe, S., & Szegedy, C. (2015). *Batch normalization: Accelerating deep network training by reducing internal covariate shift.* doi:1502.03167

Jablons, Z. (2016, April). Identifying humpback whale flukes by sequence matching of trailing edge curvature. Troy, New York: Rensselaer Polytechnic Institute.

Jones, E., Oliphant, T., Peterson, P., & Others. (2001). SciPy: Open source scientific tools for Python. Retrieved 12 19, 2018, from http://www.scipy.org

Kaggle Inc. (2018). *Humpback Whale Identification*. Retrieved 12 26, 2018, from kaggle: https://www.kaggle.com/c/humpback-whale-identification

Kaggle Inc. (2018, July 22). *Kaggle*. Retrieved from Humpback Whale Identification Challenge: https://www.kaggle.com/c/whale-categorization-playground

Katona, S. K., & Kraus, S. (1979). *Photographic identification of individual humpback whales (Megaptera novaeangliae): evaluation and analysis of the technique.* Washington, DC.: NTIS.

Katona, S. K., Harcourt, P. M., Perkins, J. S., & Kraus, S. (1980). *Humpback Whales: a catalogue of individuals identified in the western North Atlantic Ocean by means of fluke photographs.* Bar Harbour: College of the Atlantic.

Keras Team. (2018, July 25). *Github*. Retrieved from Keras: https://github.com/keras-team/keras

Kingma, D. P., & Ba, J. (2014, 12 22). *Adam: A method for stochastic optimization (2014).* doi:arXiv:1412.6980

Li, S., Jiao, J., Han, Y., & Weissman, T. (2016). *arXiv preprint arXiv.* doi:1611.01186.

Lillie, D. G. (1915). British Antartic ("Terra Nova") Expedition, 1910. *Natural History Reports, 1*(3), 85-124.

Martin, S. (2002). *The Whales' Journey: A Year in the Life of a Humpback Whale, and a Century in the History of Whaling.* Crows Nest, NSW, Australia: Allen & Unwin.

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics, 5*(4), 115-133.

Mukherjee, U. (2018, December 4). *How to handle Imbalanced Classification Problems in machine learning?* Retrieved from Analytics Vidhya: https://www.analyticsvidhya.com/blog/2017/03/imbalanced-classification-problem/

NG, A. (2017, 11 7). C4W2L03 Resnets. Deeplearning.ai. Retrieved 12 27, 2018, from https://www.youtube.com/watch?v=ZILIbUvp5lk

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., . . . Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research, 12*, 2825-2830.

Piotte, M. (2018, June 25). *Bounding box data for the whale flukes.* Retrieved from Kaggle: https://www.kaggle.com/martinpiotte/bounding-box-data-for-the-whale-flukes

Piotte, M. (2018, December 4). *Kaggle.* Retrieved from Bounding Box Model: https://www.kaggle.com/martinpiotte/bounding-box-model

Schevill, W. E., & Backus, R. H. (1960). Daily Patrol of a Megaptera. *Journal of Mammalogy, 41*(2), 279-281.

Tensorflow. (2018, July 26). *Github*. Retrieved from Tensorflow: https://github.com/tensorflow/tensorflow

Weideman, H., Jablons, Z., Holmberg, J., Flynn, K., Calambokidis, J., Tyson, R., . . . Stewart, C. (2017). Integral Curvature Representation and Matching Algorithms for Identification of Dolphins and Whales. arXiv preprint arXiv:1708.07785. Retrieved from Integral Curvature Representation and Matching Algorithms for Identification of Dolphins and Whales: http://openaccess.thecvf.com/content_ICCV_2017_workshops/papers/w41/Weideman_Integral_Curvature_Representation_ICCV_2017_paper.pdf

Weisstein, E. W. (n.d.). *Affine Transformation*. Retrieved 12 19, 2018, from Mathworld: Wolfram Web Resources: http://mathworld.wolfram.com/AffineTransformation.html

Wiesel, T. N., & Hubel, D. H. (1968). Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, *1*(195), 215-243.

Wikipedia. (2018, 7 24). *Convolutional neural network*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Convolutional_neural_network

Xavier, G., Bordes, A., & Bengio, Y. (2011). "Deep sparse rectifier neural networks." Proceedings of the. 2011. *Fourteenth international conference on artificial intelligence and statistics*, (pp. 315-323).

APPENDIX A

```
from __future__ import print_function
%matplotlib inline
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from skimage.io import imread
import numpy as np
from collections import Counter
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from PIL import Image
from PIL import ImageStat
import pickle
import tensorflow as tf
from keras.preprocessing.image import img_to_array,array_to_img
from scipy.ndimage import affine_transform
from pylab import *
from keras.utils.np_utils import to_categorical
from sklearn.preprocessing import StandardScaler
#import imagehash
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten,  BatchNormalization, Conv2D, MaxPooling2D
from keras.utils import plot_model
#import pydot
from keras.utils import np_utils
from sklearn.preprocessing import LabelEncoder,OneHotEncoder
from keras.preprocessing.image import ImageDataGenerator
import keras
from keras.callbacks import LearningRateScheduler
from keras.applications.resnet50 import ResNet50
from sklearn.preprocessing import MinMaxScaler
#from scipy.ndimage.interpolation import affine_transform
from sklearn.metrics import confusion_matrix
from keras.models import load_model
from keras.callbacks import ModelCheckpoint
import numpy as np
import tensorflow as tf
import random as rn
# The below is necessary in Python 3.2.3 onwards to
# have reproducible behavior for certain hash-based operations.
# See these references for further details:
# https://docs.python.org/3.4/using/cmdline.html#envvar-PYTHONHASHSEED
# https://github.com/keras-team/keras/issues/2280#issuecomment-306959926
import os
os.environ['PYTHONHASHSEED'] = '0'
# The below is necessary for starting Numpy generated random numbers
# in a well-defined initial state.
np.random.seed(42)
# The below is necessary for starting core Python generated random numbers
# in a well-defined state.
rn.seed(123)
# Force TensorFlow to use single thread.
# Multiple threads are a potential source of
```

```
# non-reproducible results.
# For further details, see: https://stackoverflow.com/questions/42022950/which-seeds-have-to-be-set-where-to-realize-100-reproducibility-of-training-res
session_conf = tf.ConfigProto(intra_op_parallelism_threads=1, inter_op_parallelism_threads=1)
from keras import backend as K
# The below tf.set_random_seed() will make random number generation
# in the TensorFlow backend have a well-defined initial state.
# For further details, see: https://www.tensorflow.org/api_docs/python/tf/set_random_seed
tf.set_random_seed(1234)
sess = tf.Session(graph=tf.get_default_graph(), config=session_conf)
K.set_session(sess)
print(len(os.listdir("../input/train")))
#print(len(os.listdir("../input/test")))
map = pd.read_csv("../input/train.csv")
map.head()
randomRows = map.sample(frac=1.)[:10] # randomly choose 8 rows of the .csv file
filenames = list(randomRows['Image']) # convert Image column of the dataframe to filename list
labels = list(randomRows['Id']) # convert ID column of the dataframe to filename list
images = [imread(f'../input/train/{filename}') for filename in filenames] # using formatted string for changing filenames
# create a list of arrays of rantomly chosen 10 images
figure = plt.figure(figsize=(20, 10)) # set figure size to 8 inches x 6 inches
rows = 2 # define # of rows
cols = 5 # define # of columns
for i in range(len(images)): # loop over images
    subplot = figure.add_subplot(rows, cols, i + 1) # add subplots to rows x columns figure grid
    subplot.axis('Off') #turn-off axis
    subplot.set_title(labels[i], fontsize=14) # set titles
    plt.imshow(images[i]) # show images
figure.savefig("../figures/flukes.png")
def is_gray_scale(p):
    """Adapted from https://stackoverflow.com/questions/23660929/how-to-check-whether-a-jpeg-image-is-color-or-gray-scale-using-only-python-stdli"""
    img = Image.open(f'../input/train/{p}')
    img = img.convert('RGB')
    width,height = img.size
    for i in range(width):
        for j in range(height):
            r,g,b = img.getpixel((i,j))
            if r != g != b: return False
    return True
percentage = 0 #initialize %
N = 5 #number of samples
for i in range(N):
    is_gray = [is_gray_scale(i) for i in map['Image'].sample(frac=0.01)]
    percentage = percentage + sum([i for i in is_gray]) / len([i for i in is_gray]) * 100 #add %'s of different samples up
percentage = round(percentage/N,2) #avarage the percentage sum
print(f" Nearly {percentage}% of training images are grayscaled")
catNumber = len(map['Id'].unique()) # number of unique categories are counted
print(f'Number of categories: {catNumber}')# and printed
map['Id'].value_counts() # number of images in each class is printed
le = LabelEncoder()
Ids = le.fit_transform(map['Id']) # assign ordinal levels to categorical IDs
fig = plt.figure(figsize = (8, 6))
sns.distplot(Ids, vertical = True)
plt.title('Categorical Distribution')
plt.ylabel("Ordinal Whale IDs")
```

```
plt.xlabel("PDF")
plt.show() # plot the distribution
fig.savefig("../figures/ID_dist.png")
from collections import Counter
whale_dist = Counter(map['Id'].value_counts().values)
print("\n(# of images, # of classes containing those # of images)\n")
print(sorted(whale_dist.items()))
fig = plt.figure(figsize = (8, 6))
plt.bar(range(len(whale_dist)), list(whale_dist.values())[::-1], align='center')
plt.xticks(range(len(whale_dist)), list(whale_dist.keys())[::-1])
plt.title("Number of Categories by Number of Images")
plt.xlabel('Number of Images')
plt.ylabel('Number of Categories')
plt.show()
fig.savefig('../figures/Cat_by_Number.png')
filenames = map['Image'].sample(frac=0.25)
images={filename: plt.imread(f'../input/train/{filename}') for filename in filenames}
img_sizes = Counter([value.shape[:2] for value in images.values()])
size, freq = zip(*Counter({i: v for i, v in img_sizes.items() if v > 1}).most_common(20))
fig = plt.figure(figsize=(8, 6))
plt.bar(range(len(freq)), list(freq), align='center')
plt.xticks(range(len(size)), list(size), rotation=70)
plt.title("Image size frequencies (where freq > 1)")
plt.xlabel("Image Size")
plt.ylabel("Count")
plt.show()
fig.savefig('../figures/imgSize_Dist.png')
map20 = map[map["Id"].isin(top20.ID)]
#map20.Id.unique() # check name of classes
map20.shape
map20.describe()
map20["Id"].unique()
filenames = map20['Image']#.sample(frac=0.25)
images={filename: plt.imread(f'../input/train/{filename}') for filename in filenames}
img_sizes = Counter([value.shape[:2] for value in images.values()])
size, freq = zip(*Counter({i: v for i, v in img_sizes.items() if v > 1}).most_common(20))
fig = plt.figure(figsize=(8, 6))
plt.bar(range(len(freq)), list(freq), align='center')
plt.xticks(range(len(size)), list(size), rotation=70)
plt.title("Image size frequencies (where freq > 1)")
plt.xlabel("Image Size")
plt.ylabel("Count")
plt.show()
fig.savefig('../figures/imgSize_Dist_cons.png')
with open("../input/Outliers.txt") as f1: # Open file of image names to be excluded
    outliers = f1.read().splitlines()
outlierSET = set(outliers) #convert list of names to set to avoid repetitive names
outliers = list(outlierSET) #re-convert set to list of names.
#len(outliers) # total number of outliers = 47
allExclude = list(map20[map20["Image"].isin(outliers)].Image)
toBeExcluded = [imread(f'../input/train/{filename}') for filename in allExclude]
figure = plt.figure(figsize=(20, 40)) # set figure size to 8 inches x 6 inches
rows = 1 # define # of rows
cols = 2 # define # of columns
for i in range(len(toBeExcluded)): # loop over images
    subplot = figure.add_subplot(rows, cols, i + 1) # add subplots to rows x columns figure grid
    subplot.axis('Off') #turn-off axis
    subplot.set_title(allExclude[i], fontsize=14) # set titles
```

```
    plt.imshow(toBeExcluded[i])# show images
figure.savefig("../figures/toBeExcluded2.png")
omitIndices = []
for p in allExclude:
    #print(p)
    index = list(map20["Image"]).index(p)
    omitIndices.append(index)
map20 = map20.drop(map20.index[omitIndices])
len(map20)
with open('../input/rotate.txt') as f: # Open Piotte's findings
    rotationList = f.read().splitlines()
with open("../input/extendRotate.txt") as f1: # Open mine
    allUpsdDwn = f1.read().splitlines()
totalUpsnDwn = set(rotationList).union(allUpsdDwn) #convert list of names to set to avoid repetitive names
totalUpsnDwn = list(totalUpsnDwn)
#map20[map20["Image"].isin(totalUpsnDwn)]
allRotate = list(map20[map20["Image"].isin(totalUpsnDwn)].Image)
toBeRotated = [imread(f'../input/train/{filename}') for filename in allRotate]
figure = plt.figure(figsize=(20, 40)) # set figure size to 8 inches x 6 inches
rows = 1 # define # of rows
cols = 3 # define # of columns
for i in range(len(toBeRotated)): # loop over images
    subplot = figure.add_subplot(rows, cols, i + 1) # add subplots to rows x columns figure grid
    subplot.axis('Off') #turn-off axis
    subplot.set_title(allRotate[i], fontsize=14) # set titles
    plt.imshow(toBeRotated[i])# show images
figure.savefig("../figures/toBeRotated.png")
# Read the bounding box data from the bounding box kernel (see reference above)
with open('../input/bounding-box.pickle', 'rb') as f:
    p2bb = pickle.load(f)
list(p2bb.items())[:5]
def readImage(p):
    img = Image.open(f"../input/train/{p}")
    if p in allRotate:
        img = img.rotate(180)
    #img = imread(f"data/train/{p}")
    return img
aspectRatio = 0
for p in map20.Image:
    img = readImage(p)
    width, height = img.size
    aspectRatio += width/height
aspectRatio = aspectRatio/len(map20) #take average
print("Average aspect ratio:",aspectRatio )
def crop(p):
    img_shape    = (224,224,1) # The image shape used by the model
    anisotropy   = 2.09 # The horizontal compression ratio
    margin  = 0.1 # The margin added around the bounding box to compensate for bounding box inaccuracy
    # Determine the region of the original image we want to capture based on the bounding box.
    x0,y0,x1,y1   = p2bb[p]
    # Read the image
    p = readImage(p)
    #if p in allRotate: BUG FIXED!!! no need to flip since readImage already does!
        #img = img.rotate(180)
    # Get size
    size_x,size_y = p.size
    if p in allRotate: x0, y0, x1, y1 = size_x - x1, size_y - y1, size_x - x0, size_y - y0
    dx           = x1 - x0
```

```
    dy          = y1 - y0
    x0          -= dx*margin
    x1          += dx*1.5*margin + 1
    y0          -= dy*margin
    y1          += dy*margin + 1
    if (x0 < 0):
        x0 = 0
    if (x1 > size_x):
        x1 = size_x
    if (y0 < 0):
        y0 = 0
    if (y1 > size_y):
        y1 = size_y
    dx          = x1 - x0
    dy          = y1 - y0
    if dx > dy*anisotropy:
        dy  = 0.5*(dx/anisotropy - dy)
        y0 -= dy
        y1 += dy
    else:
        dx  = 0.5*(dy*anisotropy - dx)
        x0 -= dx
        x1 += dx
    # Generate the transformation matrix
    trans = np.array([[1, 0, -0.5*img_shape[0]], [0, 1, -0.5*img_shape[1]], [0, 0, 1]])
    trans = np.dot(np.array([[(y1 - y0)/img_shape[0], 0, 0], [0, (x1 - x0)/img_shape[1], 0], [0, 0, 1]]), trans)
    trans = np.dot(np.array([[1, 0, 0.5*(y1 + y0)], [0, 1, 0.5*(x1 + x0)], [0, 0, 1]]), trans)
    # Transform to black and white and comvert to numpy array
    img   = p.convert('L') #keep 3 channel info for ResNET
    img   = img_to_array(img)#img_to_array(img)
    # Apply affine transformation
    matrix = trans[:2,:2]
    offset = trans[:2,2]
    img    = img.reshape(img.shape[:-1])#
    img    = ndimage.affine_transform(img, matrix, offset, output_shape=img_shape[:-1], order=1,
mode='constant', cval=np.average(img))
    img    = img.reshape(img_shape)
    # Normalize to zero mean and unit variance
    img  -= np.mean(img, keepdims=True)
    img  /= np.std(img, keepdims=True) + K.epsilon()
    return img
for p in allRotate:
    img = readImage(p)
    print("Original Image")
    plt.imshow(Image.open(f"../input/train/{p}"))
    plt.show()
    print("Flipped Image")
    plt.imshow(readImage(p))
    plt.show()
    print("Cropped & Reshaped Version")
    cropped =crop(p)
    #plt.imshow(cropped)
    plt.imshow(cropped.reshape(img_shape[0], img_shape[1]),cmap = matplotlib.cm.binary)
    plt.show()
with open("../input/cropFail.txt") as f: # Open file of image names to be excluded
    cropFail = list(f.read().splitlines())

#toBeExcluded = [imread(f'data/train/{filename}') for filename in allExclude]
```

```
figure = plt.figure(figsize=(20, 40)) # set figure size to 20 inches x 6 inches
rows = 9 # define # of rows
cols = 2 # define # of columns
index = 0
for i in cropFail: # loop over images
    subplot = figure.add_subplot(rows, cols, index + 1) # add subplots to rows x columns figure grid
    subplot.axis('Off') #turn-off axis
    subplot.set_title(i, fontsize=14) # set titles
    plt.imshow(readImage(i))# show images
    #plt.show()
    cropped =(crop(i))
    subplot = figure.add_subplot(rows, cols, index + 2) # add subplots to rows x columns figure grid
    subplot.axis('Off') #turn-off axis
    plt.imshow(cropped.reshape(img_shape[0], img_shape[1]),cmap = matplotlib.cm.binary)
    title = "Cropped " + i
    subplot.set_title(title, fontsize=14) # set titles
    #plt.show()
    index = index + 2
figure.savefig("../figures/cropFail.png")
omitIndices = []
for p in cropFail:
    #print(p)
    index = list(map20["Image"]).index(p)
    omitIndices.append(index)
map20 = map20.drop(map20.index[omitIndices])
len(map20)
final_top20 = pd.DataFrame(map20['Id'].value_counts().head(11))
final_top20.reset_index(inplace=True)
final_top20.columns = ['ID','Counts']
fig = plt.figure(figsize = (8, 6))
plt.title('Whale Categories with 20 or more Images')
sns.set_color_codes("pastel")
sns.barplot(x="ID", y="Counts", data=final_top20,
            label="Count")
locs, labels = plt.xticks()
plt.setp(labels, rotation=45)
plt.show()
fig.savefig('final_top20.png')
y = map20["Id"]
names = list(map20["Image"])
X = []
for i in range(len(map20)):
    imgName = names[i]
    cropped = crop(imgName)
    #cropped =to_rgb(cropped)
    X.append(cropped)
X = np.array(X)
X = X.reshape( (-1, 224, 224, 1)).astype("float32")
print(X.shape)
print(y.shape)
import pickle
with open('../input/X_gray', 'wb') as f1:
    pickle.dump(X, f1)
with open('../input/y_gray', 'wb') as f2:
    pickle.dump(y, f2)
with open ('../input/X_gray', 'rb') as f3:
    X = pickle.load(f3)
with open ('../input/y_gray', 'rb') as f4:
```

```
    y = pickle.load(f4)
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.25,stratify=y)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.15,stratify=y_train)
height = 224
width = 224
channels =1
X_train = X_train.reshape(-1,height,width,channels) # dimension (60000) is inferred by setting the first
element to -1
X_test = X_test.reshape(-1,height,width,channels)
X_val = X_val.reshape(-1,height,width,channels)
print("<3 Shape of My Heart <3 : ")
print("========================")
print("Training:", X_train.shape)
print("Testing:", X_test.shape)
print("Validation:", X_val.shape)
print(y_train.shape)
train_top20 = pd.DataFrame(y_train.value_counts())
train_top20.reset_index(inplace=True)
train_top20.columns = ['ID','Counts']
val_top20 = pd.DataFrame(y_val.value_counts())
val_top20.reset_index(inplace=True)
val_top20.columns = ['ID','Counts']
test_top20 = pd.DataFrame(y_test.value_counts())
test_top20.reset_index(inplace=True)
test_top20.columns = ['ID','Counts']
f,(ax1,ax2,ax3) = plt.subplots(3,1,sharex=True)
fig1 = sns.barplot(x="ID", y="Counts", data=train_top20, ax=ax1)
fig1.set_ylabel('Train')
fig1.set_xlabel('')
fig2 = sns.barplot(x="ID", y="Counts", data=val_top20, ax=ax2)
fig2.set_xlabel('')
fig2.set_ylabel('Validation')
fig3 = sns.barplot(x="ID", y="Counts", data=test_top20, ax=ax3)
fig3.set_ylabel('Test')
fig3.set_xlabel('')
f=plt.xticks(rotation=45)
suptitle("Categorical Distrbution of Images", fontsize=16)
#f.savefig('../figures/train_val_test_Dist.png')
y_train = LabelEncoder().fit_transform(y_train)
#y_train = OneHotEncoder().fit_transform(y_train.reshape(-1,1))
y_val = LabelEncoder().fit_transform(y_val)
#y_val = OneHotEncoder().fit_transform(y_val.reshape(-1,1))
y_test = LabelEncoder().fit_transform(y_test)
#y_test = OneHotEncoder().fit_transform(y_test.reshape(-1,1))
print(y_train.shape)
print(y_val.shape)
print(y_test.shape)
classNum = 11#len(np.unique(y_train)) # number of unique labels is counted to determine the # of classes
y_train = to_categorical(y_train, num_classes = classNum)
y_test = to_categorical(y_test, num_classes = classNum)
y_val = to_categorical(y_val, num_classes = classNum)
datagen = ImageDataGenerator(
        rotation_range=20,  # randomly rotate images in the range (degrees, 0 to 180)
        zoom_range = 0.2, # Randomly zoom image
        width_shift_range=0.2,  # randomly shift images horizontally (fraction of total width)
        height_shift_range=0.2,  # randomly shift images vertically (fraction of total height)
        horizontal_flip=True,  # randomly flip images horizontally only!
        vertical_flip=False,  #  do not randomly flip images vertically!
```

```
        fill_mode='nearest')
datagen.fit(X_train, augment=True)
inputShape = (224,224,1)
model = ResNet50(include_top = True, classes=11, input_shape = inputShape)
model.summary()
from keras.utils import plot_model
plot_model(model, to_file='../figures/models1.png')
pil_image.open('../figures/models1.png')
model.compile(loss=keras.losses.categorical_crossentropy,
            optimizer=keras.optimizers.Adam(), #instead of annealer decay = DR can be set, too
            metrics=['accuracy'])
annealer = LearningRateScheduler(lambda x: 1e-3 * 0.9 ** x)
# Fit the model
Epochs = 300 #
batchSize = 12 # number of randomly taken samples from features and labels to feed into each epoch
             # until an epoch limit is reached.
history = model.fit_generator(datagen.flow(X_train,y_train, batch_size=batchSize),
                          epochs = Epochs, validation_data = (X_val,y_val),
                          verbose = 1, steps_per_epoch=X_train.shape[0] // batchSize
                          , callbacks=[annealer])
from keras.models import load_model
model.save('../models/ResNET50.h5')  # creates a HDF5 file 'my_model.h5'
#del model  # deletes the existing model
# returns a compiled model
# identical to the previous one
#model = load_model('my_model.h5')
valLoss, valAcc = model.evaluate(X_val, y_val, verbose=0)
trainLoss, trainAcc = model.evaluate(X_train, y_train, verbose=0)
print("Validation Loss: {0:.6f}, Validation Accuracy: {1:.6f}".format(valLoss, valAcc))
print("Train Loss: {0:.6f}, Train Accuracy: {1:.6f}".format(trainLoss, trainAcc))
sns.set_color_codes("pastel")
sns.set_style("white")
sns.lineplot(x='loss', data = )
plt.plot(history.history['loss'], label = "Training Loss")
plt.plot(history.history['val_loss'], label = "Validation Loss")
plt.legend()
plt.show()
plt.plot(history.history['acc'],label = "Training Accuracy")
plt.plot(history.history['val_acc'], label = "Validation Accuracy")
plt.legend()
plt.show()
pred = model.predict(X_test) # Predict values of the test set
#y_testCat = to_categorical(y_test, num_classes = classNum)
testLoss, testAcc = model.evaluate(X_test, y_test, verbose=0)
print(testLoss,testAcc)
pred1hot = np.argmax(pred, axis=1) # Convert predicted classes to one hot vectors
y_test1hot = np.argmax(y_test, axis=1) # Convert true classes to one hot vectors
cm = confusion_matrix(y_test1hot, pred1hot) #confusion matrix
plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=True, fmt=".0f", linewidths=.5, square = True, cmap="YlGnBu");
plt.ylabel('True Labels');
plt.xlabel('Predicted Labels');

plt.title('Test Loss = %.4f, Test Accuracy = %.4f'%(testLoss, testAcc), size = 15);

#plot_confusion_matrix(confusion_mtx, classes = range(10))
with open ('../input/X_gray', 'rb') as f3:
   X = pickle.load(f3)
```

```
with open ('../input/y_gray', 'rb') as f4:
    y = pickle.load(f4)
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.25,stratify=y, random_state =42)
#X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.15,stratify=y_train)
height = 224
width = 224
channels =1
X_train = X_train.reshape(-1,height,width,channels) # dimension (60000) is inferred by setting the first
element to -1
X_test = X_test.reshape(-1,height,width,channels)
#X_val = X_val.reshape(-1,height,width,channels)
y_train = LabelEncoder().fit_transform(y_train)
#y_train = OneHotEncoder().fit_transform(y_train.reshape(-1,1))
#y_val = LabelEncoder().fit_transform(y_val)
#y_val = OneHotEncoder().fit_transform(y_val.reshape(-1,1))
y_test = LabelEncoder().fit_transform(y_test)
#y_test = OneHotEncoder().fit_transform(y_test.reshape(-1,1))
print(y_train.shape)
#print(y_val.shape)
print(y_test.shape)
classNum = 11#len(np.unique(y_train)) # number of unique labels is counted to determine the # of classes
y_train = to_categorical(y_train, num_classes = classNum)
y_test = to_categorical(y_test, num_classes = classNum)
#y_val = to_categorical(y_val, num_classes = classNum)
datagen = ImageDataGenerator(
      rotation_range=20,  # randomly rotate images in the range (degrees, 0 to 180)
      zoom_range = 0.2, # Randomly zoom image
      width_shift_range=0.2,  # randomly shift images horizontally (fraction of total width)
      height_shift_range=0.2,  # randomly shift images vertically (fraction of total height)
      horizontal_flip=True,  # randomly flip images horizontally only!
      vertical_flip=False,  #  do not randomly flip images vertically!
      fill_mode='nearest',
      zca_whitening = False) # A whitening transform of an image is a linear algebra operation that reduces
the redundancy in the matrix of pixel images. Less redundancy in the image is intended to better highlight the
structures and features in the image to the learning algorithm.
datagen.fit(X_train, augment=True)
# configure batch size and retrieve one batch of images
#os.makedirs('augmentedImages')
#for X_batch, y_batch in datagen.flow(X_train, y_train, batch_size=9, save_to_dir='../augmentedImages',
save_prefix='aug', save_format='png'):
inputShape = (224,224,1)
model = load_model('../models/ResNET50.h5')
model.compile(loss=keras.losses.categorical_crossentropy,
          optimizer=keras.optimizers.Adam(), #instead of annealer decay = DR can be set, too
          metrics=['accuracy'])
annealer = LearningRateScheduler(lambda x: 1e-3 * 0.9 ** x)
# Fit the model
Epochs = 300 #
batchSize = 12 # number of randomly taken samples from features and labels to feed into each epoch
           # until an epoch limit is reached.
history = model.fit_generator(datagen.flow(X_train,y_train, batch_size=batchSize),
                      epochs = Epochs, validation_data = (X_test,y_test),
                      verbose = 1, steps_per_epoch=X_train.shape[0] // batchSize
                      , callbacks=[annealer])
model.save('../models/ResNET50_wo_val.h5')  # creates a HDF5 file 'my_model.h5'
valLoss, valAcc = model.evaluate(X_test, y_test, verbose=0)
trainLoss, trainAcc = model.evaluate(X_train, y_train, verbose=0)
print("Validation Loss: {0:.6f}, Validation Accuracy: {1:.6f}".format(valLoss, valAcc))
```

```
print("Train Loss: {0:.6f}, Train Accuracy: {1:.6f}".format(trainLoss, trainAcc))
sns.set_color_codes("pastel")
sns.set_style("white")
#sns.lineplot(x='loss', data = )
plt.plot(history.history['loss'], label = "Training Loss")
plt.plot(history.history['val_loss'], label = "Validation Loss")
plt.legend()
plt.show()
plt.plot(history.history['acc'],label = "Training Accuracy")
plt.plot(history.history['val_acc'], label = "Validation Accuracy")
plt.legend()
plt.show()
pred = model.predict(X_test) # Predict values of the test set
#y_testCat = to_categorical(y_test, num_classes = classNum)
testLoss, testAcc = model.evaluate(X_test, y_test, verbose=0)
print(testLoss,testAcc)
pred1hot = np.argmax(pred, axis=1) # Convert predicted classes to one hot vectors
y_test1hot = np.argmax(y_test, axis=1) # Convert true classes to one hot vectors
cm = confusion_matrix(y_test1hot, pred1hot) #confusion matrix
plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=True, fmt=".0f", linewidths=.5, square = True, cmap="YlGnBu");
plt.ylabel('True Labels');
plt.xlabel('Predicted Labels');
plt.title('Test Loss = %.4f, Test Accuracy = %.4f'%(testLoss, testAcc), size = 15);
#plot_confusion_matrix(confusion_mtx, classes = range(10))
inputShape = (224,224,1)
model = load_model('../models/ResNET50_80_20_split.h5')
model.load_weights('../models/ResNET50_80_20_50batchSize_weights.h5')
model.compile(loss=keras.losses.categorical_crossentropy,
            optimizer=keras.optimizers.Adam(lr=0.01), #instead of annealer decay = DR can be set, too
            metrics=['accuracy'])
annealer = LearningRateScheduler(lambda x: 1e-3 * 0.9 ** x)
# Fit the model
Epochs = 70 #
batchSize = 50 # number of randomly taken samples from features and labels to feed into each epoch
             # until an epoch limit is reached.
# checkpoint
filepath="../models/ResNET_80_20_split_ADAM_LR0.01_weights-improvement-{epoch:02d}-
{val_acc:.2f}.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_acc', verbose=1, save_best_only=True, mode='max')
#callbacks_list = [checkpoint]
history = model.fit_generator(datagen.flow(X_train,y_train, batch_size=batchSize),
                              epochs = Epochs, validation_data = (X_test,y_test),
                              verbose = 1, steps_per_epoch=X_train.shape[0] // batchSize
                              , callbacks=[annealer, checkpoint])
model.save('../models/ResNET_80_20_split_ADAM_LR0.01.h5')  # creates a HDF5 file 'my_model.h5'
# save weights
model.save_weights('../models/ResNET_80_20_split_ADAM_LR0.01_weights.h5')
# to restore a model from a checkpoint see: https://machinelearningmastery.com/check-point-deep-learning-
models-keras/
valLoss, valAcc = model.evaluate(X_test, y_test, verbose=0)
trainLoss, trainAcc = model.evaluate(X_train, y_train, verbose=0)
print("Validation Loss: {0:.6f}, Validation Accuracy: {1:.6f}".format(valLoss, valAcc))
print("Train Loss: {0:.6f}, Train Accuracy: {1:.6f}".format(trainLoss, trainAcc))
sns.set_color_codes("pastel")
sns.set_style("white")
#sns.lineplot(x='loss', data = )
plt.plot(history.history['loss'], label = "Training Loss")
```

```
plt.plot(history.history['val_loss'], label = "Validation Loss")
plt.legend()
plt.show()
plt.plot(history.history['acc'],label = "Training Accuracy")
plt.plot(history.history['val_acc'], label = "Validation Accuracy")
plt.legend()
plt.show()
model = load_model('../models/ResNET_80_20_split_ADAM_LR0.01.h5')
model.load_weights('../models/ResNET_80_20_split_ADAM_LR0.01_weights.h5')
pred = model.predict(X_test) # Predict values of the test set
#y_testCat = to_categorical(y_test, num_classes = classNum)
testLoss, testAcc = model.evaluate(X_test, y_test, verbose=0)
print(testLoss,testAcc)
pred1hot = np.argmax(pred, axis=1) # Convert predicted classes to one hot vectors
y_test1hot = np.argmax(y_test, axis=1) # Convert true classes to one hot vectors
cm = confusion_matrix(y_test1hot, pred1hot) #confusion matrix
plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=True, fmt=".0f", linewidths=.5, square = True, cmap="YlGnBu");
plt.ylabel('True Labels');
plt.xlabel('Predicted Labels');
plt.title('Test Loss = %.4f, Test Accuracy = %.4f'%(testLoss, testAcc), size = 15);
#plot_confusion_matrix(confusion_mtx, classes = range(10))
inputShape = (224,224,1)
model = load_model('../models/ResNET_80_20_split_ADAM_LR0.01.h5')
model.load_weights('../models/ResNET_80_20_split_ADAM_LR0.01_weights.h5')
model.compile(loss=keras.losses.categorical_crossentropy,
          optimizer=keras.optimizers.Adam(lr=0.005), #instead of annealer decay = DR can be set, too
          metrics=['accuracy'])
annealer = LearningRateScheduler(lambda x: 1e-3 * 0.9 ** x)
# Fit the model
Epochs = 70 #
batchSize = 50 # number of randomly taken samples from features and labels to feed into each epoch
           # until an epoch limit is reached.
# checkpoint
filepath="../models/ResNET_80_20_split_ADAM_LR0.005_weights-improvement-{epoch:02d}-
{val_acc:.2f}.hdf5" #PREVIOUSLY WRONGLY NAMED PATH!!
checkpoint = ModelCheckpoint(filepath, monitor='val_acc', verbose=1, save_best_only=True, mode='max')
#ACCIDENTALLY WROTE 0.01 instead of 0.005 :()
#callbacks_list = [checkpoint]
history = model.fit_generator(datagen.flow(X_train,y_train, batch_size=batchSize),
                      epochs = Epochs, validation_data = (X_test,y_test),
                      verbose = 1, steps_per_epoch=X_train.shape[0] // batchSize
                      , callbacks=[annealer, checkpoint])
model.save('../models/ResNET_80_20_split_ADAM_LR0.005.h5')  # creates a HDF5 file 'my_model.h5'
# save weights
model.save_weights('../models/ResNET_80_20_split_ADAM_LR0.005_weights.h5')
# to restore a model from a checkpoint see: https://machinelearningmastery.com/check-point-deep-learning-
models-keras/
valLoss, valAcc = model.evaluate(X_test, y_test, verbose=0)
trainLoss, trainAcc = model.evaluate(X_train, y_train, verbose=0)
print("Validation Loss: {0:.6f}, Validation Accuracy: {1:.6f}".format(valLoss, valAcc))
print("Train Loss: {0:.6f}, Train Accuracy: {1:.6f}".format(trainLoss, trainAcc))
sns.set_color_codes("pastel")
sns.set_style("white")
#sns.lineplot(x='loss', data = )
plt.plot(history.history['loss'], label = "Training Loss")
plt.plot(history.history['val_loss'], label = "Validation Loss")
plt.legend()
```

```
plt.show()
plt.plot(history.history['acc'],label = "Training Accuracy")
plt.plot(history.history['val_acc'], label = "Validation Accuracy")
plt.legend()
plt.show()
model = load_model('../models/ResNET_80_20_split_ADAM_LR0.005.h5')
model.load_weights('../models/ResNET_80_20_split_ADAM_LR0.005_weights.h5')
pred = model.predict(X_test) # Predict values of the test set
#y_testCat = to_categorical(y_test, num_classes = classNum)
testLoss, testAcc = model.evaluate(X_test, y_test, verbose=0)
print(testLoss,testAcc)
pred1hot = np.argmax(pred, axis=1) # Convert predicted classes to one hot vectors
y_test1hot = np.argmax(y_test, axis=1) # Convert true classes to one hot vectors
cm = confusion_matrix(y_test1hot, pred1hot) #confusion matrix
plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=True, fmt=".0f", linewidths=.5, square = True, cmap="YlGnBu");
plt.ylabel('True Labels');
plt.xlabel('Predicted Labels');
plt.title('Test Loss = %.4f, Test Accuracy = %.4f'%(testLoss, testAcc), size = 15);
#plot_confusion_matrix(confusion_mtx, classes = range(10))
inputShape = (224,224,1)
model = load_model('../models/ResNET50_90_10_split.h5')
model.load_weights('../models/ResNET50_90_10_weights.h5')
model.compile(loss=keras.losses.categorical_crossentropy,
            optimizer=keras.optimizers.Adam(), #instead of annealer decay = DR can be set, too
            metrics=['accuracy'])
annealer = LearningRateScheduler(lambda x: 1e-3 * 0.9 ** x)
# Fit the model
Epochs = 70 #
batchSize = 50 # number of randomly taken samples from features and labels to feed into each epoch
              # until an epoch limit is reached.
# checkpoint
filepath="../models/ResNET_80_20_split_LR0.001-{epoch:02d}-{val_acc:.2f}.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_acc', verbose=1, save_best_only=True, mode='max')
#callbacks_list = [checkpoint]
history = model.fit_generator(datagen.flow(X_train,y_train, batch_size=batchSize),
                          epochs = Epochs, validation_data = (X_test,y_test),
                          verbose = 1, steps_per_epoch=X_train.shape[0] // batchSize
                          , callbacks=[annealer, checkpoint])
model.save('../models/ResNET50_80_20_split_LR0.001.h5')  # creates a HDF5 file 'my_model.h5'
# save weights
model.save_weights('../models/ResNET50_80_20_LR0.001_weights.h5')
# to restore a model from a checkpoint see: https://machinelearningmastery.com/check-point-deep-learning-
models-keras/
valLoss, valAcc = model.evaluate(X_test, y_test, verbose=0)
trainLoss, trainAcc = model.evaluate(X_train, y_train, verbose=0)
print("Validation Loss: {0:.6f}, Validation Accuracy: {1:.6f}".format(valLoss, valAcc))
print("Train Loss: {0:.6f}, Train Accuracy: {1:.6f}".format(trainLoss, trainAcc))
sns.set_color_codes("pastel")
sns.set_style("white")
#sns.lineplot(x='loss', data = )
plt.plot(history.history['loss'], label = "Training Loss")
plt.plot(history.history['val_loss'], label = "Validation Loss")
plt.legend()
plt.show()
plt.plot(history.history['acc'],label = "Training Accuracy")
plt.plot(history.history['val_acc'], label = "Validation Accuracy")
plt.legend()
```

```
plt.show()
pred = model.predict(X_test) # Predict values of the test set
#y_testCat = to_categorical(y_test, num_classes = classNum)
testLoss, testAcc = model.evaluate(X_test, y_test, verbose=0)
print(testLoss,testAcc)
pred1hot = np.argmax(pred, axis=1) # Convert predicted classes to one hot vectors
y_test1hot = np.argmax(y_test, axis=1) # Convert true classes to one hot vectors
cm = confusion_matrix(y_test1hot, pred1hot) #confusion matrix
plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=True, fmt=".0f", linewidths=.5, square = True, cmap="YlGnBu");
plt.ylabel('True Labels');
plt.xlabel('Predicted Labels');
plt.title('Test Loss = %.4f, Test Accuracy = %.4f'%(testLoss, testAcc), size = 15);
#plot_confusion_matrix(confusion_mtx, classes = range(10))
def get_lr_metric(optimizer):
   def lr(y_true, y_pred):
      return optimizer.lr
   return lr
inputShape = (224,224,1)
model = load_model('../models/ResNET_80_20_split_ADAM_LR0.005.h5')
model.load_weights('../models/ResNET_80_20_split_ADAM_LR0.005_weights.h5')
opt = keras.optimizers.Adam(lr=0.004323416)
lr_metric = get_lr_metric(opt)
model.compile(loss=keras.losses.categorical_crossentropy,
          optimizer=keras.optimizers.Adam(lr=0.004323416), #instead of annealer decay = DR can be set, too
          metrics=['accuracy', lr_metric])
annealer = LearningRateScheduler(lambda x: 1e-3 * 0.9 ** x)
# Fit the model
Epochs = 70 #
batchSize = 50 # number of randomly taken samples from features and labels to feed into each epoch
            # until an epoch limit is reached.
# checkpoint
filepath="../models/ResNET_80_20_split_ADAM_LR0.0043_weights-improvement-{epoch:02d}-
{val_acc:.2f}.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_acc', verbose=1, save_best_only=True, mode='max')
#callbacks_list = [checkpoint]
history = model.fit_generator(datagen.flow(X_train,y_train, batch_size=batchSize),
                        epochs = Epochs, validation_data = (X_test,y_test),
                        verbose = 1, steps_per_epoch=X_train.shape[0] // batchSize
                        , callbacks=[annealer, checkpoint])

…..

from keras.callbacks import Callback
import keras.backend as K
import numpy as np

class SGDRScheduler(Callback):
   '''Cosine annealing learning rate scheduler with periodic restarts.
   # Usage
      ```python
         schedule = SGDRScheduler(min_lr=1e-5,
                              max_lr=1e-2,
                              steps_per_epoch=np.ceil(epoch_size/batch_size),
                              lr_decay=0.9,
                              cycle_length=5,
                              mult_factor=1.5)
         model.fit(X_train, Y_train, epochs=100, callbacks=[schedule])
```
```

```
    ```
Arguments
    min_lr: The lower bound of the learning rate range for the experiment.
    max_lr: The upper bound of the learning rate range for the experiment.
    steps_per_epoch: Number of mini-batches in the dataset. Calculated as `np.ceil(epoch_size/batch_size)`.
    lr_decay: Reduce the max_lr after the completion of each cycle.
              Ex. To reduce the max_lr by 20% after each cycle, set this value to 0.8.
    cycle_length: Initial number of epochs in a cycle.
    mult_factor: Scale epochs_to_restart after each full cycle completion.
References
Blog post: jeremyjordan.me/nn-learning-rate
    Original paper: http://arxiv.org/abs/1608.03983
'''
def __init__(self,
             min_lr,
             max_lr,
             steps_per_epoch,
             lr_decay=1,
             cycle_length=10,
             mult_factor=2):

    self.min_lr = min_lr
    self.max_lr = max_lr
    self.lr_decay = lr_decay
    self.batch_since_restart = 0
    self.next_restart = cycle_length
    self.steps_per_epoch = steps_per_epoch
    self.cycle_length = cycle_length
    self.mult_factor = mult_factor
    self.history = {}

def clr(self):
    '''Calculate the learning rate.'''
    fraction_to_restart = self.batch_since_restart / (self.steps_per_epoch * self.cycle_length)
    lr = self.min_lr + 0.5 * (self.max_lr - self.min_lr) * (1 + np.cos(fraction_to_restart * np.pi))
    return lr

def on_train_begin(self, logs={}):
    '''Initialize the learning rate to the minimum value at the start of training.'''
    logs = logs or {}
    K.set_value(self.model.optimizer.lr, self.max_lr)

def on_batch_end(self, batch, logs={}):
    '''Record previous batch statistics and update the learning rate.'''
    logs = logs or {}
    self.history.setdefault('lr', []).append(K.get_value(self.model.optimizer.lr))
    for k, v in logs.items():
        self.history.setdefault(k, []).append(v)
    self.batch_since_restart += 1
    K.set_value(self.model.optimizer.lr, self.clr())

def on_epoch_end(self, epoch, logs={}):
    '''Check for end of current cycle, apply restarts when necessary.'''
    if epoch + 1 == self.next_restart:
        self.batch_since_restart = 0
        self.cycle_length = np.ceil(self.cycle_length * self.mult_factor)
        self.next_restart += self.cycle_length
        self.max_lr *= self.lr_decay
```
```

```
            self.best_weights = self.model.get_weights()
    def on_train_end(self, logs={}):
        '''Set weights to the values from the end of the most recent cycle for best performance.'''
        self.model.set_weights(self.best_weights)

inputShape = (224,224,1)
model = load_model('../models/ResNET50_80_20_split.h5')
model.load_weights('../models/ResNET50_80_20_50batchSize_weights.h5')
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.SGD(lr=0.01, momentum=0.9, decay=5e-4, nesterov=True), #instead of
annealer decay = DR can be set, too
              metrics=['accuracy'])

# Fit the model
Epochs = 70 #
batchSize = 50 # number of randomly taken samples from features and labels to feed into each epoch
               # until an epoch limit is reached.
# cosine annealer reference: https://arxiv.org/abs/1608.03983
#annealer = LearningRateScheduler(lambda x: 1e-3 * 0.9 ** x)
schedule = SGDRScheduler(min_lr=1e-4,
                         max_lr=1e-2,
                         steps_per_epoch = X_train.shape[0] // batchSize,#np.ceil(epoch_size/batch_size),
                         lr_decay=0.9,
                         cycle_length=5,
                         mult_factor=1.5)
# checkpoint
filepath="../models/ResNET_SGD_LR0.01_weights-improvement-{epoch:02d}-{val_acc:.2f}.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_acc', verbose=1, save_best_only=True, mode='max')

history = model.fit_generator(datagen.flow(X_train,y_train, batch_size=batchSize),
                              epochs = Epochs, validation_data = (X_test,y_test),
                              verbose = 1, steps_per_epoch=X_train.shape[0] // batchSize
                              , callbacks=[schedule, checkpoint])
model.save('../models/ResNET50_SGD_LR001.h5')  # creates a HDF5 file 'my_model.h5'
# save weights
model.save_weights('../models/ResNET50_SGD_LR001_weights.h5')
valLoss, valAcc = model.evaluate(X_test, y_test, verbose=0)
trainLoss, trainAcc = model.evaluate(X_train, y_train, verbose=0)
print("Validation Loss: {0:.6f}, Validation Accuracy: {1:.6f}".format(valLoss, valAcc))
print("Train Loss: {0:.6f}, Train Accuracy: {1:.6f}".format(trainLoss, trainAcc))
sns.set_color_codes("pastel")
sns.set_style("white")
#sns.lineplot(x='loss', data = )
plt.plot(history.history['loss'], label = "Training Loss")
plt.plot(history.history['val_loss'], label = "Validation Loss")
plt.legend()
plt.show()
plt.plot(history.history['acc'],label = "Training Accuracy")
plt.plot(history.history['val_acc'], label = "Validation Accuracy")
plt.legend()
plt.show()
pred = model.predict(X_test) # Predict values of the test set
#y_testCat = to_categorical(y_test, num_classes = classNum)
testLoss, testAcc = model.evaluate(X_test, y_test, verbose=0)
print(testLoss,testAcc)
pred1hot = np.argmax(pred, axis=1) # Convert predicted classes to one hot vectors
y_test1hot = np.argmax(y_test, axis=1) # Convert true classes to one hot vectors
cm = confusion_matrix(y_test1hot, pred1hot) #confusion matrix
```

```
plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=True, fmt=".0f", linewidths=.5, square = True, cmap="YlGnBu");
plt.ylabel('True Labels');
plt.xlabel('Predicted Labels');
plt.title('Test Loss = %.4f, Test Accuracy = %.4f'%(testLoss, testAcc), size = 15);
#plot_confusion_matrix(confusion_mtx, classes = range(10))
```

APPENDIX B

Layer (type)	Output Shape	Param #	Connected to
input_5 (InputLayer)	(None, 224, 224, 1)	0	
zero_padding2d_4 (ZeroPadding2D	(None, 230, 230, 1)	0	input_5[0][0]
conv1 (Conv2D)	(None, 112, 112, 64)	3200	zero_padding2d_4[0][0]
bn_conv1 (BatchNormalization)	(None, 112, 112, 64)	256	conv1[0][0]
activation_197 (Activation)	(None, 112, 112, 64)	0	bn_conv1[0][0]
max_pooling2d_5 (MaxPooling2D)	(None, 55, 55, 64)	0	activation_197[0][0]
res2a_branch2a (Conv2D)	(None, 55, 55, 64)	4160	max_pooling2d_5[0][0]
bn2a_branch2a (BatchNormalizati	(None, 55, 55, 64)	256	res2a_branch2a[0][0]
activation_198 (Activation)	(None, 55, 55, 64)	0	bn2a_branch2a[0][0]
res2a_branch2b (Conv2D)	(None, 55, 55, 64)	36928	activation_198[0][0]
bn2a_branch2b (BatchNormalizati	(None, 55, 55, 64)	256	res2a_branch2b[0][0]
activation_199 (Activation)	(None, 55, 55, 64)	0	bn2a_branch2b[0][0]
res2a_branch2c (Conv2D)	(None, 55, 55, 256)	16640	activation_199[0][0]
res2a_branch1 (Conv2D)	(None, 55, 55, 256)	16640	max_pooling2d_5[0][0]
bn2a_branch2c (BatchNormalizati	(None, 55, 55, 256)	1024	res2a_branch2c[0][0]
bn2a_branch1 (BatchNormalizatio	(None, 55, 55, 256)	1024	res2a_branch1[0][0]
add_65 (Add)	(None, 55, 55, 256)	0	bn2a_branch2c[0][0] bn2a_branch1[0][0]
activation_200 (Activation)	(None, 55, 55, 256)	0	add_65[0][0]
res2b_branch2a (Conv2D)	(None, 55, 55, 64)	16448	activation_200[0][0]
bn2b_branch2a (BatchNormalizati	(None, 55, 55, 64)	256	res2b_branch2a[0][0]
activation_201 (Activation)	(None, 55, 55, 64)	0	bn2b_branch2a[0][0]
res2b_branch2b (Conv2D)	(None, 55, 55, 64)	36928	activation_201[0][0]
bn2b_branch2b (BatchNormalizati	(None, 55, 55, 64)	256	res2b_branch2b[0][0]
activation_202 (Activation)	(None, 55, 55, 64)	0	bn2b_branch2b[0][0]
res2b_branch2c (Conv2D)	(None, 55, 55, 256)	16640	activation_202[0][0]
bn2b_branch2c (BatchNormalizati	(None, 55, 55, 256)	1024	res2b_branch2c[0][0]
add_66 (Add)	(None, 55, 55, 256)	0	bn2b_branch2c[0][0] activation_200[0][0]
activation_203 (Activation)	(None, 55, 55, 256)	0	add_66[0][0]
res2c_branch2a (Conv2D)	(None, 55, 55, 64)	16448	activation_203[0][0]
bn2c_branch2a (BatchNormalizati	(None, 55, 55, 64)	256	res2c_branch2a[0][0]
activation_204 (Activation)	(None, 55, 55, 64)	0	bn2c_branch2a[0][0]
res2c_branch2b (Conv2D)	(None, 55, 55, 64)	36928	activation_204[0][0]
bn2c_branch2b (BatchNormalizati	(None, 55, 55, 64)	256	res2c_branch2b[0][0]
activation_205 (Activation)	(None, 55, 55, 64)	0	bn2c_branch2b[0][0]
res2c_branch2c (Conv2D)	(None, 55, 55, 256)	16640	activation_205[0][0]
bn2c_branch2c (BatchNormalizati	(None, 55, 55, 256)	1024	res2c_branch2c[0][0]
add_67 (Add)	(None, 55, 55, 256)	0	bn2c_branch2c[0][0] activation_203[0][0]

activation_206 (Activation)	(None, 55, 55, 256)	0	add_67[0][0]
res3a_branch2a (Conv2D)	(None, 28, 28, 128)	32896	activation_206[0][0]
bn3a_branch2a (BatchNormalizati	(None, 28, 28, 128)	512	res3a_branch2a[0][0]
activation_207 (Activation)	(None, 28, 28, 128)	0	bn3a_branch2a[0][0]
res3a_branch2b (Conv2D)	(None, 28, 28, 128)	147584	activation_207[0][0]
bn3a_branch2b (BatchNormalizati	(None, 28, 28, 128)	512	res3a_branch2b[0][0]
activation_208 (Activation)	(None, 28, 28, 128)	0	bn3a_branch2b[0][0]
res3a_branch2c (Conv2D)	(None, 28, 28, 512)	66048	activation_208[0][0]
res3a_branch1 (Conv2D)	(None, 28, 28, 512)	131584	activation_206[0][0]
bn3a_branch2c (BatchNormalizati	(None, 28, 28, 512)	2048	res3a_branch2c[0][0]
bn3a_branch1 (BatchNormalizatio	(None, 28, 28, 512)	2048	res3a_branch1[0][0]
add_68 (Add)	(None, 28, 28, 512)	0	bn3a_branch2c[0][0] bn3a_branch1[0][0]
activation_209 (Activation)	(None, 28, 28, 512)	0	add_68[0][0]
res3b_branch2a (Conv2D)	(None, 28, 28, 128)	65664	activation_209[0][0]
bn3b_branch2a (BatchNormalizati	(None, 28, 28, 128)	512	res3b_branch2a[0][0]
activation_210 (Activation)	(None, 28, 28, 128)	0	bn3b_branch2a[0][0]
res3b_branch2b (Conv2D)	(None, 28, 28, 128)	147584	activation_210[0][0]
bn3b_branch2b (BatchNormalizati	(None, 28, 28, 128)	512	res3b_branch2b[0][0]
activation_211 (Activation)	(None, 28, 28, 128)	0	bn3b_branch2b[0][0]
res3b_branch2c (Conv2D)	(None, 28, 28, 512)	66048	activation_211[0][0]
bn3b_branch2c (BatchNormalizati	(None, 28, 28, 512)	2048	res3b_branch2c[0][0]
add_69 (Add)	(None, 28, 28, 512)	0	bn3b_branch2c[0][0] activation_209[0][0]
activation_212 (Activation)	(None, 28, 28, 512)	0	add_69[0][0]
res3c_branch2a (Conv2D)	(None, 28, 28, 128)	65664	activation_212[0][0]
bn3c_branch2a (BatchNormalizati	(None, 28, 28, 128)	512	res3c_branch2a[0][0]
activation_213 (Activation)	(None, 28, 28, 128)	0	bn3c_branch2a[0][0]
res3c_branch2b (Conv2D)	(None, 28, 28, 128)	147584	activation_213[0][0]
bn3c_branch2b (BatchNormalizati	(None, 28, 28, 128)	512	res3c_branch2b[0][0]
activation_214 (Activation)	(None, 28, 28, 128)	0	bn3c_branch2b[0][0]
res3c_branch2c (Conv2D)	(None, 28, 28, 512)	66048	activation_214[0][0]
bn3c_branch2c (BatchNormalizati	(None, 28, 28, 512)	2048	res3c_branch2c[0][0]
add_70 (Add)	(None, 28, 28, 512)	0	bn3c_branch2c[0][0] activation_212[0][0]
activation_215 (Activation)	(None, 28, 28, 512)	0	add_70[0][0]
res3d_branch2a (Conv2D)	(None, 28, 28, 128)	65664	activation_215[0][0]
bn3d_branch2a (BatchNormalizati	(None, 28, 28, 128)	512	res3d_branch2a[0][0]
activation_216 (Activation)	(None, 28, 28, 128)	0	bn3d_branch2a[0][0]
res3d_branch2b (Conv2D)	(None, 28, 28, 128)	147584	activation_216[0][0]
bn3d_branch2b (BatchNormalizati	(None, 28, 28, 128)	512	res3d_branch2b[0][0]
activation_217 (Activation)	(None, 28, 28, 128)	0	bn3d_branch2b[0][0]
res3d_branch2c (Conv2D)	(None, 28, 28, 512)	66048	activation_217[0][0]

```
bn3d_branch2c (BatchNormalizati (None, 28, 28, 512)  2048        res3d_branch2c[0][0]
__________________________________________________________________________________________________
add_71 (Add)                    (None, 28, 28, 512)  0           bn3d_branch2c[0][0]
                                                                 activation_215[0][0]
__________________________________________________________________________________________________
activation_218 (Activation)     (None, 28, 28, 512)  0           add_71[0][0]
__________________________________________________________________________________________________
res4a_branch2a (Conv2D)         (None, 14, 14, 256)  131328      activation_218[0][0]
__________________________________________________________________________________________________
bn4a_branch2a (BatchNormalizati (None, 14, 14, 256)  1024        res4a_branch2a[0][0]
__________________________________________________________________________________________________
activation_219 (Activation)     (None, 14, 14, 256)  0           bn4a_branch2a[0][0]
__________________________________________________________________________________________________
res4a_branch2b (Conv2D)         (None, 14, 14, 256)  590080      activation_219[0][0]
__________________________________________________________________________________________________
bn4a_branch2b (BatchNormalizati (None, 14, 14, 256)  1024        res4a_branch2b[0][0]
__________________________________________________________________________________________________
activation_220 (Activation)     (None, 14, 14, 256)  0           bn4a_branch2b[0][0]
__________________________________________________________________________________________________
res4a_branch2c (Conv2D)         (None, 14, 14, 1024) 263168      activation_220[0][0]
__________________________________________________________________________________________________
res4a_branch1 (Conv2D)          (None, 14, 14, 1024) 525312      activation_218[0][0]
__________________________________________________________________________________________________
bn4a_branch2c (BatchNormalizati (None, 14, 14, 1024) 4096        res4a_branch2c[0][0]
__________________________________________________________________________________________________
bn4a_branch1 (BatchNormalizatio (None, 14, 14, 1024) 4096        res4a_branch1[0][0]
__________________________________________________________________________________________________
add_72 (Add)                    (None, 14, 14, 1024) 0           bn4a_branch2c[0][0]
                                                                 bn4a_branch1[0][0]
__________________________________________________________________________________________________
activation_221 (Activation)     (None, 14, 14, 1024) 0           add_72[0][0]
__________________________________________________________________________________________________
res4b_branch2a (Conv2D)         (None, 14, 14, 256)  262400      activation_221[0][0]
__________________________________________________________________________________________________
bn4b_branch2a (BatchNormalizati (None, 14, 14, 256)  1024        res4b_branch2a[0][0]
__________________________________________________________________________________________________
activation_222 (Activation)     (None, 14, 14, 256)  0           bn4b_branch2a[0][0]
__________________________________________________________________________________________________
res4b_branch2b (Conv2D)         (None, 14, 14, 256)  590080      activation_222[0][0]
__________________________________________________________________________________________________
bn4b_branch2b (BatchNormalizati (None, 14, 14, 256)  1024        res4b_branch2b[0][0]
__________________________________________________________________________________________________
activation_223 (Activation)     (None, 14, 14, 256)  0           bn4b_branch2b[0][0]
__________________________________________________________________________________________________
res4b_branch2c (Conv2D)         (None, 14, 14, 1024) 263168      activation_223[0][0]
__________________________________________________________________________________________________
bn4b_branch2c (BatchNormalizati (None, 14, 14, 1024) 4096        res4b_branch2c[0][0]
__________________________________________________________________________________________________
add_73 (Add)                    (None, 14, 14, 1024) 0           bn4b_branch2c[0][0]
                                                                 activation_221[0][0]
__________________________________________________________________________________________________
activation_224 (Activation)     (None, 14, 14, 1024) 0           add_73[0][0]
__________________________________________________________________________________________________
res4c_branch2a (Conv2D)         (None, 14, 14, 256)  262400      activation_224[0][0]
__________________________________________________________________________________________________
bn4c_branch2a (BatchNormalizati (None, 14, 14, 256)  1024        res4c_branch2a[0][0]
__________________________________________________________________________________________________
activation_225 (Activation)     (None, 14, 14, 256)  0           bn4c_branch2a[0][0]
__________________________________________________________________________________________________
res4c_branch2b (Conv2D)         (None, 14, 14, 256)  590080      activation_225[0][0]
__________________________________________________________________________________________________
bn4c_branch2b (BatchNormalizati (None, 14, 14, 256)  1024        res4c_branch2b[0][0]
__________________________________________________________________________________________________
activation_226 (Activation)     (None, 14, 14, 256)  0           bn4c_branch2b[0][0]
__________________________________________________________________________________________________
res4c_branch2c (Conv2D)         (None, 14, 14, 1024) 263168      activation_226[0][0]
__________________________________________________________________________________________________
bn4c_branch2c (BatchNormalizati (None, 14, 14, 1024) 4096        res4c_branch2c[0][0]
__________________________________________________________________________________________________
add_74 (Add)                    (None, 14, 14, 1024) 0           bn4c_branch2c[0][0]
                                                                 activation_224[0][0]
__________________________________________________________________________________________________
activation_227 (Activation)     (None, 14, 14, 1024) 0           add_74[0][0]
__________________________________________________________________________________________________
res4d_branch2a (Conv2D)         (None, 14, 14, 256)  262400      activation_227[0][0]
__________________________________________________________________________________________________
bn4d_branch2a (BatchNormalizati (None, 14, 14, 256)  1024        res4d_branch2a[0][0]
__________________________________________________________________________________________________
activation_228 (Activation)     (None, 14, 14, 256)  0           bn4d_branch2a[0][0]
__________________________________________________________________________________________________
res4d_branch2b (Conv2D)         (None, 14, 14, 256)  590080      activation_228[0][0]
__________________________________________________________________________________________________
bn4d_branch2b (BatchNormalizati (None, 14, 14, 256)  1024        res4d_branch2b[0][0]
__________________________________________________________________________________________________
```

activation_229 (Activation)	(None, 14, 14, 256)	0	bn4d_branch2b[0][0]
res4d_branch2c (Conv2D)	(None, 14, 14, 1024)	263168	activation_229[0][0]
bn4d_branch2c (BatchNormalizati	(None, 14, 14, 1024)	4096	res4d_branch2c[0][0]
add_75 (Add)	(None, 14, 14, 1024)	0	bn4d_branch2c[0][0] activation_227[0][0]
activation_230 (Activation)	(None, 14, 14, 1024)	0	add_75[0][0]
res4e_branch2a (Conv2D)	(None, 14, 14, 256)	262400	activation_230[0][0]
bn4e_branch2a (BatchNormalizati	(None, 14, 14, 256)	1024	res4e_branch2a[0][0]
activation_231 (Activation)	(None, 14, 14, 256)	0	bn4e_branch2a[0][0]
res4e_branch2b (Conv2D)	(None, 14, 14, 256)	590080	activation_231[0][0]
bn4e_branch2b (BatchNormalizati	(None, 14, 14, 256)	1024	res4e_branch2b[0][0]
activation_232 (Activation)	(None, 14, 14, 256)	0	bn4e_branch2b[0][0]
res4e_branch2c (Conv2D)	(None, 14, 14, 1024)	263168	activation_232[0][0]
bn4e_branch2c (BatchNormalizati	(None, 14, 14, 1024)	4096	res4e_branch2c[0][0]
add_76 (Add)	(None, 14, 14, 1024)	0	bn4e_branch2c[0][0] activation_230[0][0]
activation_233 (Activation)	(None, 14, 14, 1024)	0	add_76[0][0]
res4f_branch2a (Conv2D)	(None, 14, 14, 256)	262400	activation_233[0][0]
bn4f_branch2a (BatchNormalizati	(None, 14, 14, 256)	1024	res4f_branch2a[0][0]
activation_234 (Activation)	(None, 14, 14, 256)	0	bn4f_branch2a[0][0]
res4f_branch2b (Conv2D)	(None, 14, 14, 256)	590080	activation_234[0][0]
bn4f_branch2b (BatchNormalizati	(None, 14, 14, 256)	1024	res4f_branch2b[0][0]
activation_235 (Activation)	(None, 14, 14, 256)	0	bn4f_branch2b[0][0]
res4f_branch2c (Conv2D)	(None, 14, 14, 1024)	263168	activation_235[0][0]
bn4f_branch2c (BatchNormalizati	(None, 14, 14, 1024)	4096	res4f_branch2c[0][0]
add_77 (Add)	(None, 14, 14, 1024)	0	bn4f_branch2c[0][0] activation_233[0][0]
activation_236 (Activation)	(None, 14, 14, 1024)	0	add_77[0][0]
res5a_branch2a (Conv2D)	(None, 7, 7, 512)	524800	activation_236[0][0]
bn5a_branch2a (BatchNormalizati	(None, 7, 7, 512)	2048	res5a_branch2a[0][0]
activation_237 (Activation)	(None, 7, 7, 512)	0	bn5a_branch2a[0][0]
res5a_branch2b (Conv2D)	(None, 7, 7, 512)	2359808	activation_237[0][0]
bn5a_branch2b (BatchNormalizati	(None, 7, 7, 512)	2048	res5a_branch2b[0][0]
activation_238 (Activation)	(None, 7, 7, 512)	0	bn5a_branch2b[0][0]
res5a_branch2c (Conv2D)	(None, 7, 7, 2048)	1050624	activation_238[0][0]
res5a_branch1 (Conv2D)	(None, 7, 7, 2048)	2099200	activation_236[0][0]
bn5a_branch2c (BatchNormalizati	(None, 7, 7, 2048)	8192	res5a_branch2c[0][0]
bn5a_branch1 (BatchNormalizatio	(None, 7, 7, 2048)	8192	res5a_branch1[0][0]
add_78 (Add)	(None, 7, 7, 2048)	0	bn5a_branch2c[0][0] bn5a_branch1[0][0]
activation_239 (Activation)	(None, 7, 7, 2048)	0	add_78[0][0]
res5b_branch2a (Conv2D)	(None, 7, 7, 512)	1049088	activation_239[0][0]
bn5b_branch2a (BatchNormalizati	(None, 7, 7, 512)	2048	res5b_branch2a[0][0]
activation_240 (Activation)	(None, 7, 7, 512)	0	bn5b_branch2a[0][0]

res5b_branch2b (Conv2D)	(None, 7, 7, 512)	2359808	activation_240[0][0]
bn5b_branch2b (BatchNormalizati	(None, 7, 7, 512)	2048	res5b_branch2b[0][0]
activation_241 (Activation)	(None, 7, 7, 512)	0	bn5b_branch2b[0][0]
res5b_branch2c (Conv2D)	(None, 7, 7, 2048)	1050624	activation_241[0][0]
bn5b_branch2c (BatchNormalizati	(None, 7, 7, 2048)	8192	res5b_branch2c[0][0]
add_79 (Add)	(None, 7, 7, 2048)	0	bn5b_branch2c[0][0] activation_239[0][0]
activation_242 (Activation)	(None, 7, 7, 2048)	0	add_79[0][0]
res5c_branch2a (Conv2D)	(None, 7, 7, 512)	1049088	activation_242[0][0]
bn5c_branch2a (BatchNormalizati	(None, 7, 7, 512)	2048	res5c_branch2a[0][0]
activation_243 (Activation)	(None, 7, 7, 512)	0	bn5c_branch2a[0][0]
res5c_branch2b (Conv2D)	(None, 7, 7, 512)	2359808	activation_243[0][0]
bn5c_branch2b (BatchNormalizati	(None, 7, 7, 512)	2048	res5c_branch2b[0][0]
activation_244 (Activation)	(None, 7, 7, 512)	0	bn5c_branch2b[0][0]
res5c_branch2c (Conv2D)	(None, 7, 7, 2048)	1050624	activation_244[0][0]
bn5c_branch2c (BatchNormalizati	(None, 7, 7, 2048)	8192	res5c_branch2c[0][0]
add_80 (Add)	(None, 7, 7, 2048)	0	bn5c_branch2c[0][0] activation_242[0][0]
activation_245 (Activation)	(None, 7, 7, 2048)	0	add_80[0][0]
avg_pool (AveragePooling2D)	(None, 1, 1, 2048)	0	activation_245[0][0]
flatten_3 (Flatten)	(None, 2048)	0	avg_pool[0][0]
fc2 (Dense)	(None, 11)	22539	flatten_3[0][0]

Total params: 23,603,979
Trainable params: 23,550,859
Non-trainable params: 53,120