Image Analysis and Computer Vision

Topic 7 Sea lion Classification

2017/2018

POLITECNICO
MILANO 1863

Computer Science and Engineering

Instructor : Vincenzo Caglioti

Supervisor: Giacomo Boracchi
Diego Carrera

Team: Zhou Yinan 10551242
Wang Xuan 10549075
Sofia Mitoulaki 10597553

Contents

1. Problem Formation

1.1 Kaggle Competition Introduction

This project is based on a previous Kaggle competition : NOAA Fisheries Steller Sea Lion Population Count. The details of the competition can be found on the website : https://www.kaggle.com/c/noaa-fisheries-steller-sea-lion-population-count

Steller sea lions in the western Aleutian Islands have declined 94 percent in the last 30 years. Having accurate population estimates enables biologists to better understand factors that may be contributing to the lack of recovery of Steller in this area. Currently, it takes biologists up to four months to count sea lions from the thousands of images NOAA Fisheries collects each year. The results of these counts are time-sensitive.

In this competition, Kagglers are invited to develop algorithms which accurately count the number of sea lions in aerial photographs. There are totally five types of sea lions : adult males, subadult males, adult females, juveniles and puppies. The dataset provided by Kaggle is quite large, around 100 GB. However, the training data is far less than the test data. There are about 950 training images and 18000 test images. In the training dataset, for each image, we have 2 versions: the original one and the other one with sea lions labeled in different color dots. Here is a sample of training data information, the numbers are the sea lion counts:

Train_id	Adult_males	Subadult_males	Adult_females	Juveniles	Pups
0	62	12	486	42	344
1	2	20	0	12	0
2	2	0	38	20	0

original image

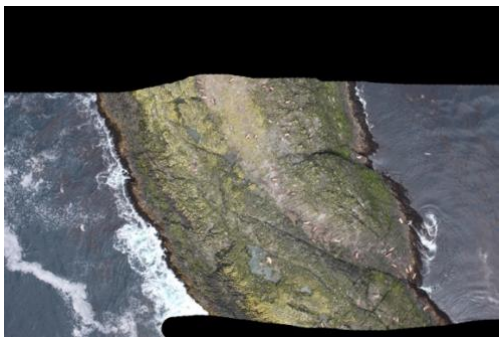


image with colored label

The performance is measured using RMSE from the human-labelled ground truth, averaged over different sea lion types.

1.2 Challenges and Solutions

First of all, let's view the problem in a nutshell. What we have are 950 image pairs : (original image, color-labeled image). We need to develop an algorithm so that we can learn the patterns in the images and when we input a new image, we can output the number of sea lions in it.

Our solution is to treat it as a classification problem. Classification is one of the main topic in computer vision field, and there are many approaches to solve it: from basic machine learning techniques like support vector machines to deep learning models like convolutional neural networks. In order to convert sea lion counting problem into sea lion classification problem, we do the following:

1. Extract small patches from images to construct train set. Each patch is either a background or contains a sea lion in middle. Converted samples in the train set have the form: (patch, label).
2. Develop an algorithm to classify a patch into one of the following six classes: adult males, adult females, subadult males, juveniles, puppies, backgrounds.
3. For a new image, first use a slide window to split it into many patches. Then use the algorithm we learned to classify each patch. Finally add up all the classification results to output sea lion counts for each sea lion class.

1.2.1 Patch Extraction and Train Set Construction

The dataset we currently have are (original image, color-labeled image) pairs. We want to convert it into (patch, label) pairs. Then we can use this as training set to develop our algorithm. We do it in the following procedure:

1. Take the color-labeled image and use blob detection to get the coordinates of the color dot.
2. From the coordinates of the colored-dot, extract patch from the original image. Suppose the patch size we define is WxW and the dot coordinate we have is (x0, y0). Then the coordinates for the four corners of the patch in the original images are: (x0±W/2, y0±W/2).
3. Use the RGB values of the color dot to distinguish between different colors. This gives us the information of sea lion class. For example, red dots imply adult-males.

1.2.2 Classification Algorithm

In this project, we experiment with two classification architectures: logistic regression and convolutional neural networks.

Logistic regression is a standard machine learning algorithm for classification. The advantage of this algorithm is that it is fast to train and fast for inference. It is usually used as a baseline. In order for logistic regression to work, we manually design some features for each patch. As we have noticed, there are a lot of background patches in each image, like the sea or bare grounds. They are quite easy to detect because the sea patch should have high values in Blue Channel and low values in Red and Green channels. So we design a simple feature pattern for each patch : average RGB values. For example:

background patch average RGB values

Since this feature pattern is quite simple, it can't distinguish between different sea lion types. So we decide to use logistic regression as a binary classifier, telling if a patch contains a sea lion or not. Then for those patches containing a sea lion, we can use some more complex algorithms like CNN to classify the sea lion type.

Convolutional Neural Networks (CNN) is a deep learning model which achieves the state of art performance for image classification. The details of CNN are introduced in the Background section. The advantage of CNN is the high accuracy but the disadvantages are obvious too. It has a lot of parameters to train, when the architecture is deep, the number of parameters can be millions. We can not train a deep CNN architecture using our own PC with CPU. Usually we need to use GPU to speed up the process.

In this project, we experiment with logistic regression and convolutional neural network. More overview, we examine the idea of fully convolutional network and transfer learning.

1.2.3 Some Challenges

The main challenges in this competition are the following:

- Each image is quite large, with high resolution. But useful information is little, since sea lions are small. Most areas are sea and bare grounds.

  Solution: By extracting patches from images, we use the patch to train the algorithm instead of using the whole image. Also we can control the number of background patches by discarding some of them.

- Compared with test dataset, we have far less training data.

  Solution: We can use some data augmentation techniques to enrich our training set. For example, by flipping the patch or rotating the patch, etc.

- We do not know where the sea lion is, so some kind of object localization is needed.

  Solution: By converting sea lion counting problem into sea lion classification problem. We only need to extract patches from each image and classify the patches. Thus we can simplify the problem and avoid object localization.

- Some data pre-processing to extract coordinates of sea lions.

  Solution: Blob detection.

- The original images and color labeled images are not exactly the same. In order to provide more accurate labels, human experts removed some areas using black regions. (as we can see in section 1.1). We need to deal with these black areas so that sea lions and backgrounds in black regions are not considered.

  Solution: By using a mask to filter out black regions. We detect where the black regions are, this is trivial because the pixel values are all zero in RBG channels. Then when we do blob detection, we do not consider these regions.

2. Background

In this section, we briefly talk about the classification algorithms we used in our project.

2.1 Logistic Regression

Logistic regression is a log linear model for classification. The basic version of logistic regression is used for binary classification problem. More specifically, we want to learn a

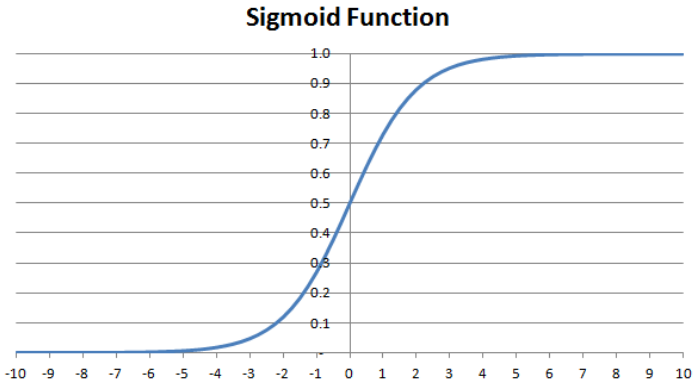
function $h_\theta(x)$, which calculates the probability of x being classified as positive label. Since it is a binary classification problem, the probability of negative label is $1 - h_\theta(x)$.

In order for the output to be a probability, it should be between 0 and 1. Logistic regression uses sigmoid function to squeeze the output.

$$h_\theta(x) = \frac{1}{1 + e^{-\theta * x}}$$

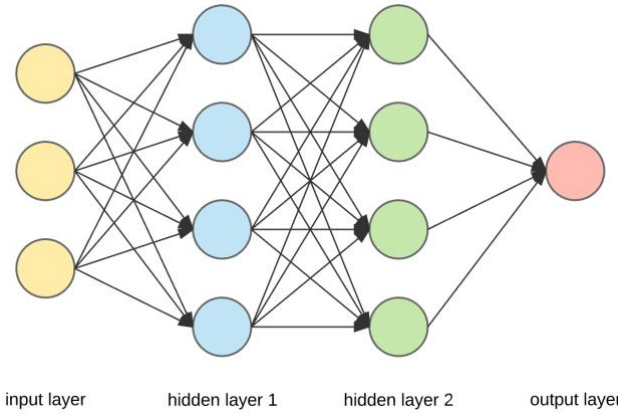
By using maximum likelihood, we can learn the optimal parameter θ through our training data set. When a new sample arrives, we calculate $h_\theta(x)$. If it is larger than a predefined threshold (0.5 for example) then we classify it as a positive sample.

Also note that we can modify the threshold for different purposes. If we increase the threshold, it is to make our prediction more reliable. We only classify it as positive if we are quite sure about it. On the other hand, when we decrease the threshold, we tend to make more positive predictions. This means that we don't want to omit positive data. In a word, we increase the threshold to optimize precision and decrease threshold to optimize recall.



2.2 Convolutional Neural Network (CNN)

Before introducing CNN, let's have a brief look at what is a neural network.

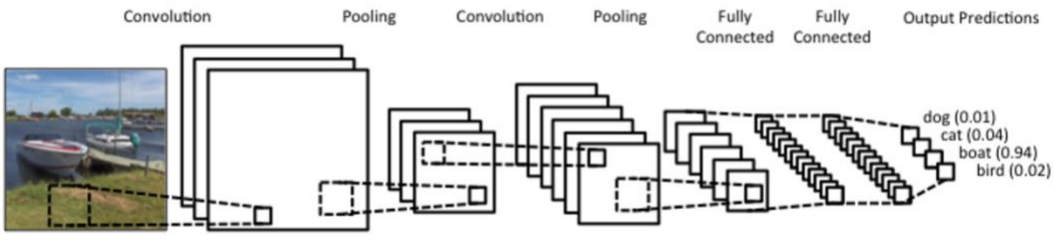


The above figure is a fully connected three layer neural network. The circles in the hidden and output layers are called neurons. The lines connected neurons are called weights. In the deep learning model, we want to learn the weights. The basic way to learn the weights are:

1. Define loss functions to evaluate the output
2. Input the network with train data
3. Compute the loss
4. Update the weights with backpropagation
5. Repeat for several epochs until loss does not decrease

Convolutional Neural Networks (CNNs) are a category of neural networks, their convolutional layers alternate with subsampling layers, reminiscent of simple and complex cells in the primary visual cortex. CNNs differentiate in how convolutional and subsampling layers are implemented and how the networks are trained.

There are four main operations in the CNNs: Convolution, Non Linearity, Pooling, Fully connection. These operations are the basic building blocks of every Convolutional Neural Network and represent the state of the art.



https://medium.com/@Aj.Cheng/convolutional-neural-network-d9f69e473feb (date of access 10-5-2018)

Convolution Layer is the core building block of CNN that does most of the computational

heavy lifting. Convolution uses filters to extract spatial information within a certain local area. To train a CNN architecture is actually to learn the filter weights. Each convolutional layer will have k filters (or kernels), each of size n x n x q where n is smaller than the dimension of the image and q is the number of input channels. The output of convolution operation is : m x m x k, where m is defined by

$$m = \frac{W - n + 2 * P}{S}$$

W = input size, P = padding, S = stride

After convolution operation, a non-linear operation is performed. The purpose of introducing non-linearity is to increase the expression power of CNN. One popular non-linearity is called ReLU which stands for Rectified Linear Unit. Its output is given by:

$$output = \max(0, input)$$

ReLU is an element wise operation (pixel-by-pixel) and replaces all negative pixel values in the feature map by zero. ReLU activation function is currently the most used activation function in CNN architecture. Compared with other activation functions like tanh and sigmoid, ReLU does not have the issue of saturation and usually leads to faster convergence.

Pooling reduces the dimensions of each feature map while retaining the most important information. Pooling can be of different types: Max, Average, Sum etc. In case of Max Pooling, we define a spatial neighborhood (for example, a 2×2 window) and take the largest element from the rectified feature map within that window. Besides taking the largest element we could also take the average (Average Pooling) or sum of all elements in that window.

Fully connected layers connect every neuron in one layer to every neuron in another layer. The output from the convolutional and pooling layers represent high-level features of the input image. The purpose of the Fully Connected layer is to use these features for classifying the input image into various classes based on the training dataset (Simard et al., 2003; Shuiwand et al., 2013).

The basic block for a CNN architecture is Conv -> ReLU-> Pooling. After several these blocks, the ideal case is that we extract the features of the image, and then we can exploit these features by using fully connected layers.

Compared with standard deep learning models, CNN is suitable for image analysis. Convolution and pooling layers greatly reduce the number of parameters to learn while maintaining the most useful spatial information.

3. Implementation Details

3.1 Overview of our CNN architecture

Our CNN architecture is composed of 5 layers, details are described in the following chart. For simplicity, the batch size is assumed to be 1. After each convolution operation, a reLu activation function is implemented. The fifth layer in our architecture is a fully connected layer. A softmax layer is added to the last. A softmax function is defined as:

$$y = \frac{e^{x_i}}{\sum e^{x_j}}$$

Layer	Operation	Input size	Filter size	Padding	Stride	Output size
Layer 1	Convolution	96*96*3	5*5*8	0	1	92*92*8
	MaxPooling	92*92*8	2*2	0	2	46*46*8
Layer 2	Convolution	46*46*8	3*3*5	0	1	44*44*5
	MaxPooling	44*44*5	2*2	0	2	22*22*5
Layer 3	Convolution	22*22*5	3*3*5	0	1	20*20*5
	MaxPooling	20*20*5	2*2	0	2	10*10*5
Layer 4	Convolution	10*10*5	3*3*10	0	1	8*8*10
	MaxPooling	8*8*10	2*2	0	2	4*4*10
Layer 5	Fully Connected	160	/	/	/	256
	Fully Connected	256	/	/	/	6

The reason to use softmax function is to squeeze the output to 0 and 1. Normally, when we deal with classification problem, we want to have a probability output, thus we can decide the class using the largest probability value.

3.2 Dataset Construction

3.2.1 Overview

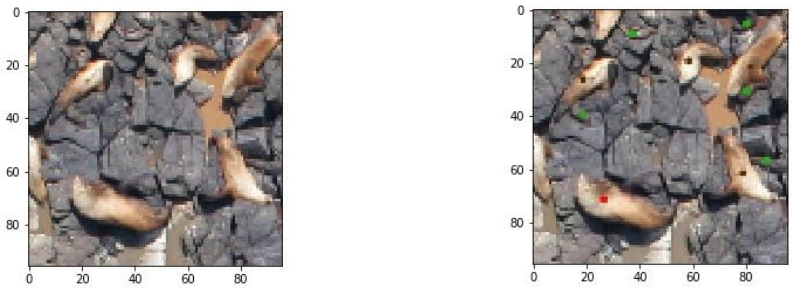
We have totally 947 training images. Different images may have different sizes but all the sizes are around (4500, 3000, 3). Thus the image is quite large and each image takes 5MB to 10 MB. Feeding the whole image into our CNN is not a feasible choice, since it requires huge computation and RAM. So our approach is first separating all the images into small patches and use the patches to train our CNN. Each patch is of the size (96, 96, 3). We chose to use this size because it is not too big but big enough to contain an adult-male sea lion which is biggest among all the sea lion types.

Each image has two copies: the original one and one with dotted sealions. In the dotted image, each sea lion class is labelled with a different color, for example adult-males are labelled in red, pups are labelled in green, etc. Here is a sample:

Original image

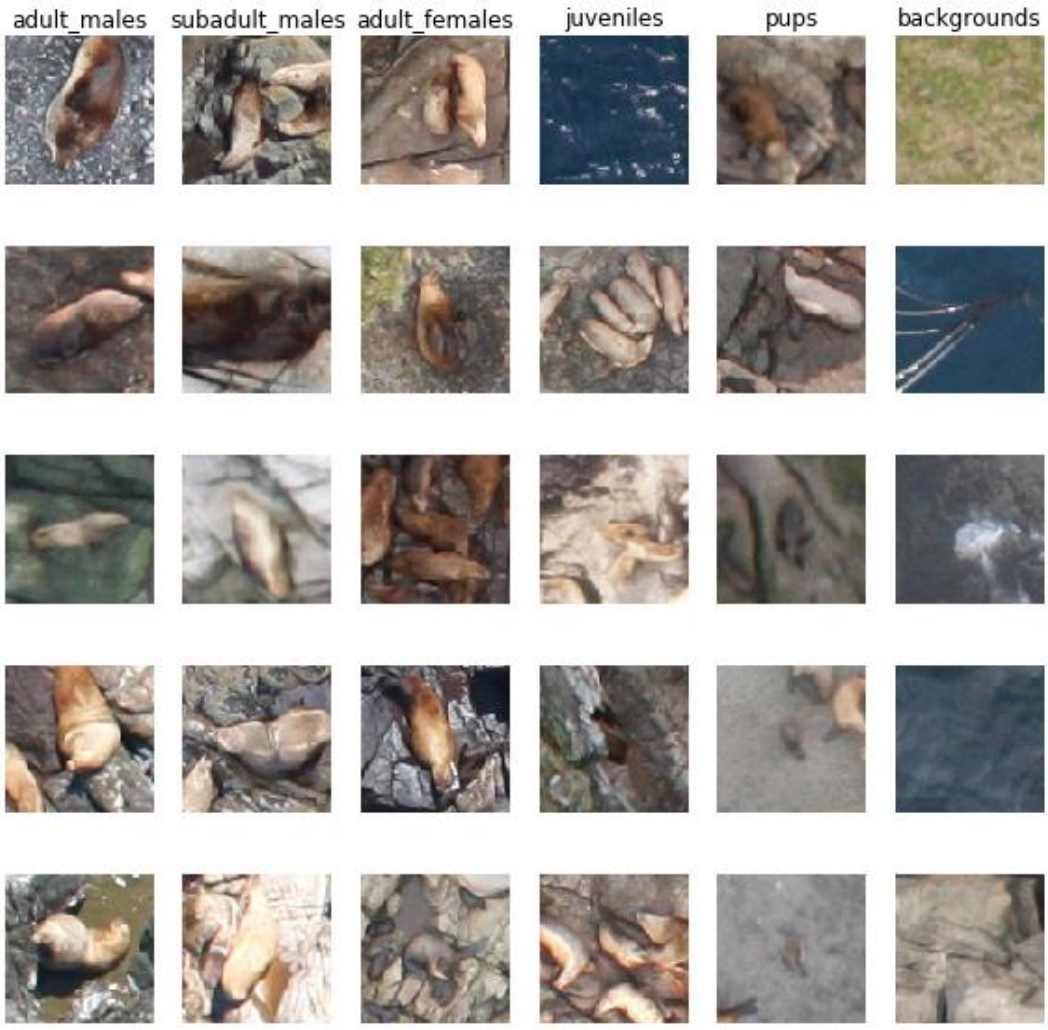
Dotted image

A zoomed in version:

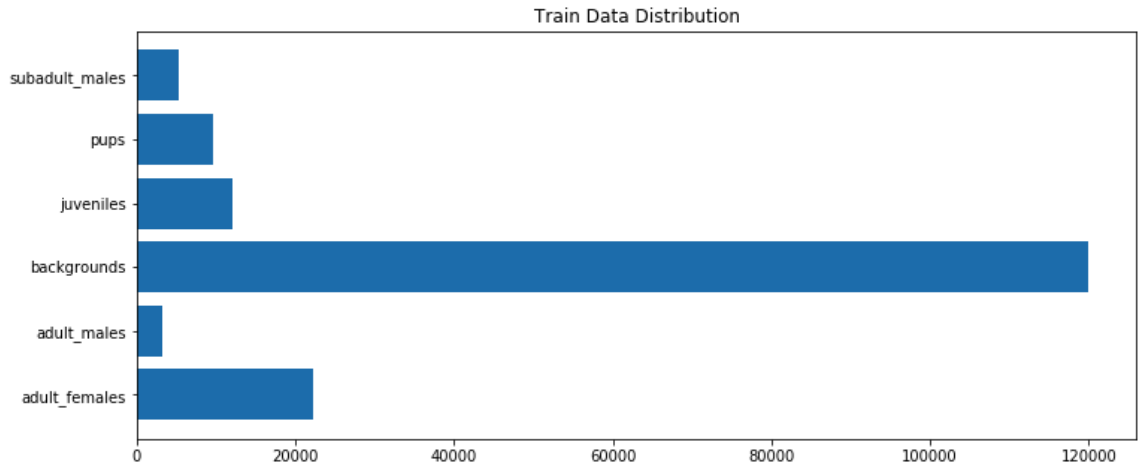


There are totally 5 classes : adult-males, subadult-males, adult-females, juveniles, pups. The patches are extracted as described in section 1.2.1.

Some training patches samples are presented below:



We have totally 172748 training patches extracted. Note that the dataset is not balanced. We have a lot more background patches than sea lion patches. For sea lions patches, adult-females are most common sea lion class. The data distribution is described here:



3.2.2 Balance Data Set

In this project, we use two metrics to evaluate our algorithm performance:

1. Accuracy:

$$accuracy = \frac{number\ of\ correct\ predictions}{number\ of\ total\ samples}$$

2. Cross Entropy Loss

$$cross\ enropty\ loss = \sum_i -logp_i$$

$$where\ p_i\ is\ defined\ by\ the\ softmax\ function.$$

Since the distribution of our patches is highly unbalanced, we need to balance the dataset otherwise the evaluation metrics will be too optimistic. For an extreme case, if our algorithm predicts every patch as a background patch, then it is a useless algorithm but yet it will give us high evaluation scores.

There are two ways to balance the dataset:

1. Up/Down sampling

By up sampling or down sampling, we change the distribution of our dataset. For example,

if we have two classes A and B with distribution 1:10. We can perform up sampling for class A to make it ten times more.

2. Assign different class weights

This method works in the loss function. As mentioned before, if we have a highly unbalanced dataset, in order to minimize the loss, our algorithm will learn to predict everything into the most common class. In order to avoid it, we can assign different class weights to different classes. For example, if we predict a sea lion patch wrong, then it will give much higher loss than we predict a background patch wrong.

In our algorithm, we experimented with both methods. They achieved similar results, and we used “different class weights” as our final decision.

3.2.3 Data Augmentation

Recall that in the Kaggle competition, we only have 970 training images, but 18000 testing images. In order to increase the model performance, we can do some data augmentation. Data augmentation is a common technique to enrich training set. In machine learning and deep learning field, more data usually means higher performance. It may not be that easy to create new data for a general problem, but for image classification problem, data augmentation is straight forward. Take an image, we can rotate, flip, zoom or shift. The following code is an example for data augmentation using Python in Keras framework.

```
trainset = ImageDataGenerator(
    rotation_range=180,
    width_shift_range=0.2,
    height_shift_range=0.2,
    fill_mode="nearest",
    horizontal_flip=True,
    rescale=1. / 255
)
```

When we load a patch into memory, it can randomly perform some operations. Each patch can rotate from 0 to 180 degrees. They can also be shifted or flipped. The last rescale parameter is done for normalization purpose. Since each pixel value of our input is between 0-255, we normalize it to 0 – 1.

3.2 Deep Learning Framework

In this project, we choose to use Keras with Tensorflow as backend. Keras is a high level

deep learning library in python. It provides API for quick experiments. The following code shows how to create a 5 layer CNN architecture which is used in our project.

```
model = Sequential()
# First layer
model.add(Convolution2D(8, (5, 5), activation='relu', padding='valid', input_shape=(96, 96, 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Second layer
model.add(Convolution2D(5, (3, 3), activation='relu', padding='valid'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Third layer
model.add(Convolution2D(5, (3, 3), activation='relu', padding='valid'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Fourth layer
model.add(Convolution2D(10, (3, 3), activation='relu', padding='valid'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Fully connected layer
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(6, activation='softmax'))
```

The model summary:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 92, 92, 8)         608
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 46, 46, 8)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 44, 44, 5)         365
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 22, 22, 5)         0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 20, 20, 5)         230
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 10, 10, 5)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 8, 8, 10)          460
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 4, 4, 10)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 160)               0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               41216
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 1542
=================================================================
Total params: 44,421
Trainable params: 44,421
Non-trainable params: 0
_________________________________________________________________
```

As we can see that there are totally 44421 parameters to train.

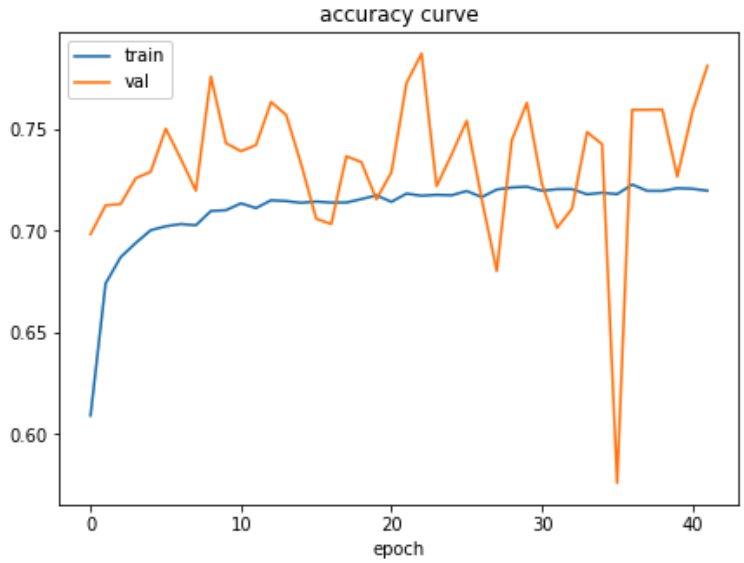
3.3 Train, Valid, Test

We follow the traditional deep learning procedure. We split all the patches into 3: train, validation and test. Train data is used to learn the weights in our network. Valid data is used to illustrate the performance for the current epoch and as an indicator for early stopping, for example when the valid loss does not decrease we can stop training. Also the loss curve of train and valid data can be used to check whether there is overfitting. The test data is used for final evaluation of our CNN. For these 947 images, we select the images with index 750 – 947 as our local test data. They are not seen by the network in training and validation phase.

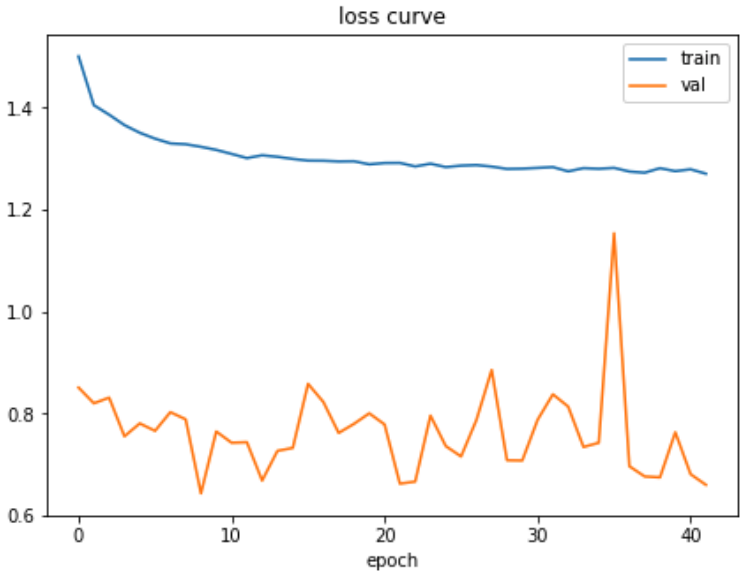
Gradient descent is the basic idea used in deep learning architecture, it defines how we could update the weights in order to reduce the loss. Gradient descent has many versions of improvements, like SGD, SGD with momentum, AdaGrad, RmsProp, and Adam. Compared with the naïve gradient descent, they use different techniques to have faster convergence.

During training, we use Adam optimizer and batch size 60. The loss function is cross-entropy. Each batch data is randomly selected from dataset and loaded into memory. Data augmentation described above is performed each time batch data is generated.

We trained our network for 40 epochs. Each epoch our network will see all the training patches in a random order.



The train accuracy curve is quite steady, but the validation accuracy fluctuates quite lot. However the general trend of accuracy score is increasing. One note about accuracy is that it is not a good metric for evaluating the performance. We do not assign weights to the accuracy metric. So if we classify every patch as a background patch, we will get a high accuracy score. The accuracy metric just gives a feeling about how the network is working. For the purpose of learning and evaluation, we should use the weighted loss metric.



As you may notice, the validation loss is less than the train loss. The reason for this is we assign class weights in the training process but not in the validation process. We could add class weights to the validation process too, but we decided not to, since we want to use validation data as a simulation for the real cases. The validation score indicates the performance of our algorithm when we encounter unbalanced data(more background patches).

After each epoch, we save the model as model_each. If we get a higher valid score than the previous one, we also save this model as model_best. After the training, we could retrieve these models for testing.

4. Experiments and Result Analysis

4.1 Logistic Regression

We plan to use logistic regression as a pre step for our CNN architecture. Logistic regression acts like a binary classifier to decide whether a patch contains a sea lion or not. If it contains a sea lion, then we'll use CNN to distinguish the sea lion types. In order for this idea to work, we need to make logistic regression to have a high recall, because we do not want to discard any possible sea lions. So we set the threshold to be low.

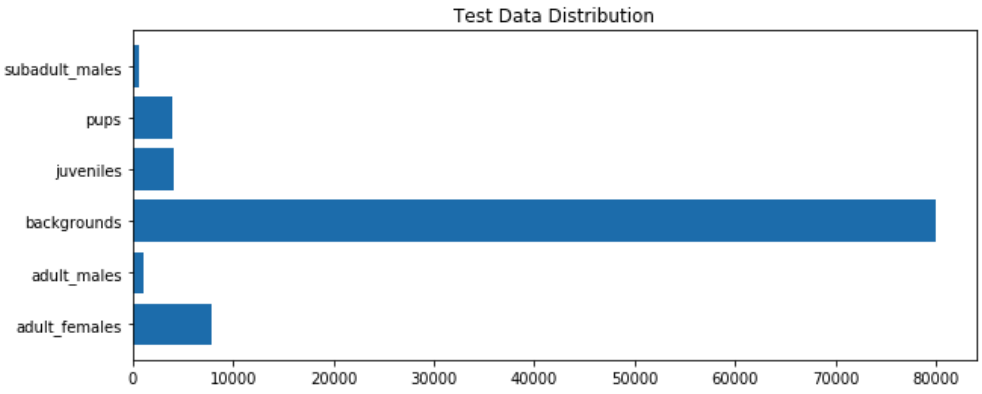
In practice, we use RGB values as sample features, and threshold 0.2. The performance is not so satisfying. We get a recall score of 0.98 and precision score 0.12. Compared with the all positive baseline, if we predict everything as a positive patch (containing sea lion), the recall score is 1.0 and precision score is 0.09. Our logistic regression algorithm performs slightly better than the baseline.

The reason for this is that RGB values alone may not be sufficient to distinguish between all background patches and sea lion patches. The features we create are still too simple for this binary classification task. The solution for this is to create more features. However, during the project we find that embed background patch into the CNN architecture achieves better results and thus we decide to use CNN as a six class classifier(sea lion + background) and discard the idea of using logistic regression as a pre-step.

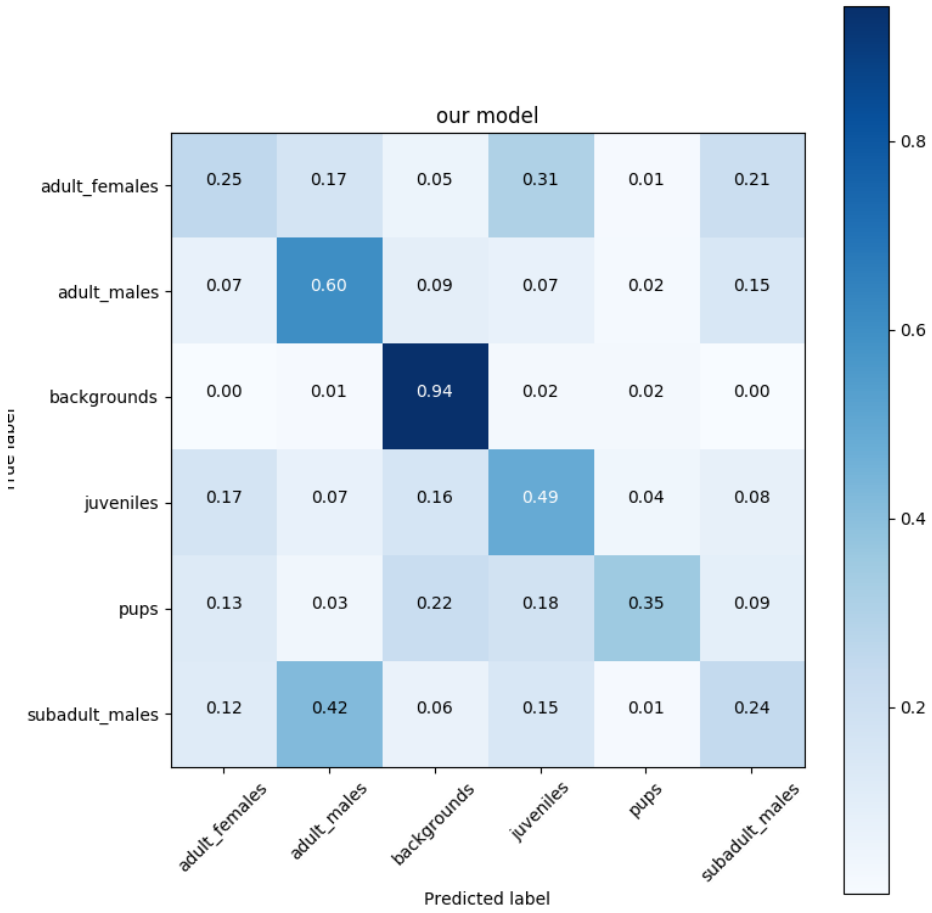
4.2 Classification Score

For a 6-class classification problem, our best model achieves 0.52 loss and 0.83 accuracy. The high accuracy is due to the fact that we have a lot of background patches in our test set, and our model is very good at classifying background patch from sea lion patch. Test

set data distribution is:



As illustrated in the figure, test set data distribution is the same as our train set. In order to analysis the performance for multi-class classification problem, we draw the confusion



matrix.

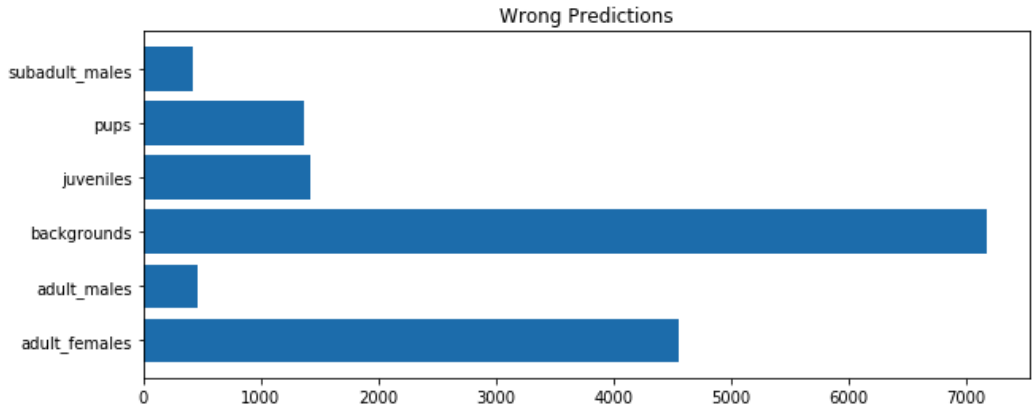
The left side is the true label and the bottom side is our predicted label. To analysis the precision score, we can look at the column data. For the recall sore, we can analysis the row.

The diagonal data are the true positive rates. As we can see, our model is very good at determining the background patches. In our opinion, this happens because of two reasons: 1. Background patches are easier to detect. For example, the sea water patch is basically blue and very few sea lions stay in the water. 2. We have a lot of background patches to train. So it is reasonable to classify the background with high precision.

4.3 Error Analysis

During our experiment, we found that several classes are very hard to distinguish between each other. For example, pups are very difficult to be distinguished from background. Because they are very small and look like a rock. Adult-females and juveniles are difficult to distinguish because they share a similar size. In the following, we check out all the wrong predictions we made.

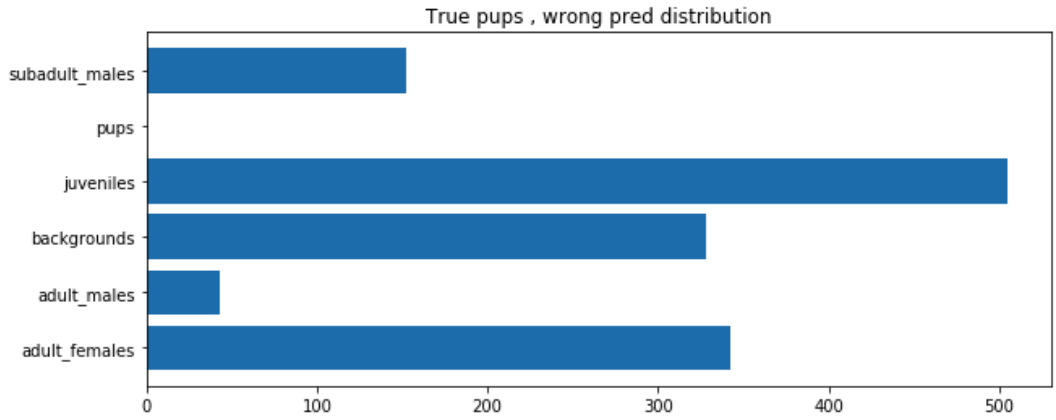
We have totally 97666 patches for testing. In all these predictions, 15412 are wrong. Here is the wrong patch class distribution:



As we can see, the wrong patch prediction has the same data distribution with our training dataset. It means that our model actually learns how to make predictions in order to minimize the error.

Here we show the distribution of the wrong predictions for each sea lion.

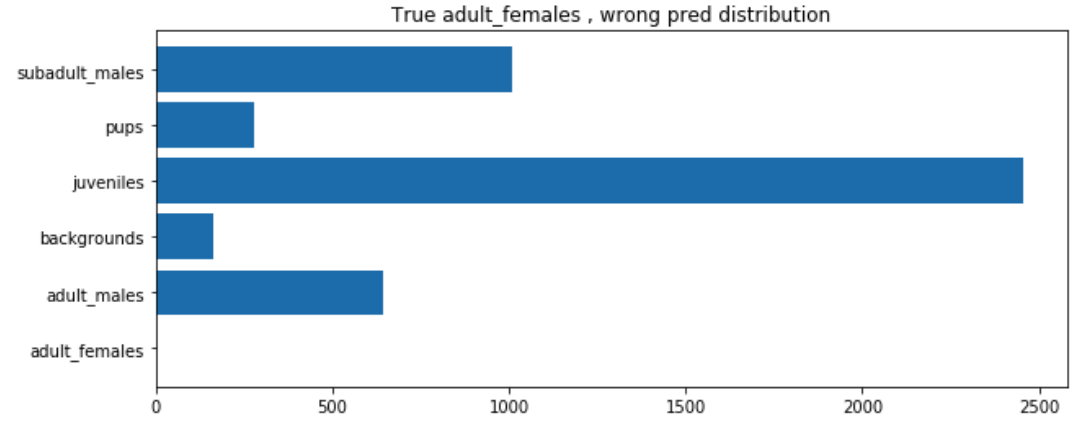
1.True Pups



The pups are most likely to be classified as juveniles. This result is not consistent with our observations that pups are like backgrounds. The reason is that pups seem to appear together with its mother or other sea lions. Pups are not likely to appear alone, far away from other sea lions. Also for juveniles, they seem like to stick together. Since our patch does not contain a single sea lion, so this surrounding sea lions are treated as additional information for a certain sea lion class. Recall that in section 1.2.1, we described how to extract patches, starting from a colored dot, we draw a square window around it, thus each patch may not contain a single sea lion. For example:

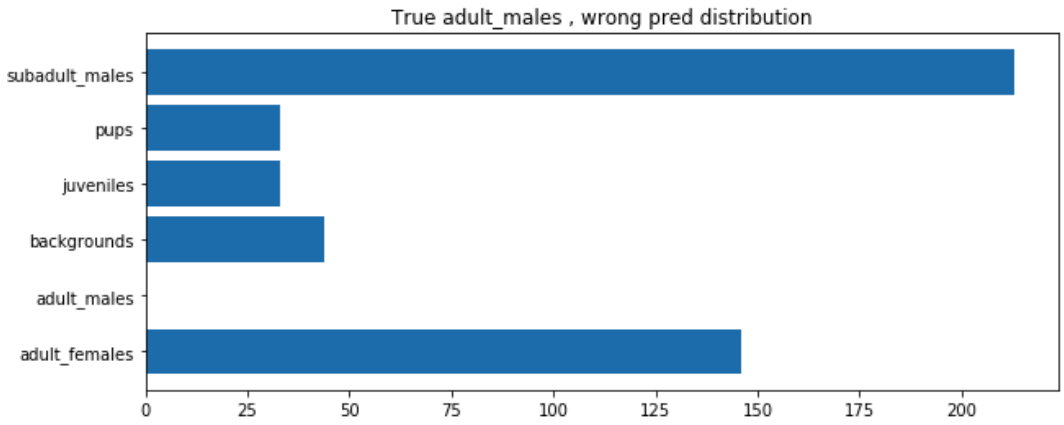
The right patch contains a single sea lion, but the left patch contains multiple sea lions. When we do convolution operation, the surrounding areas are treated as additional information, thus all the surrounding sea lions can be seen as helper information. This may also be a problem, and we'll describe it later.

2.True adult_females



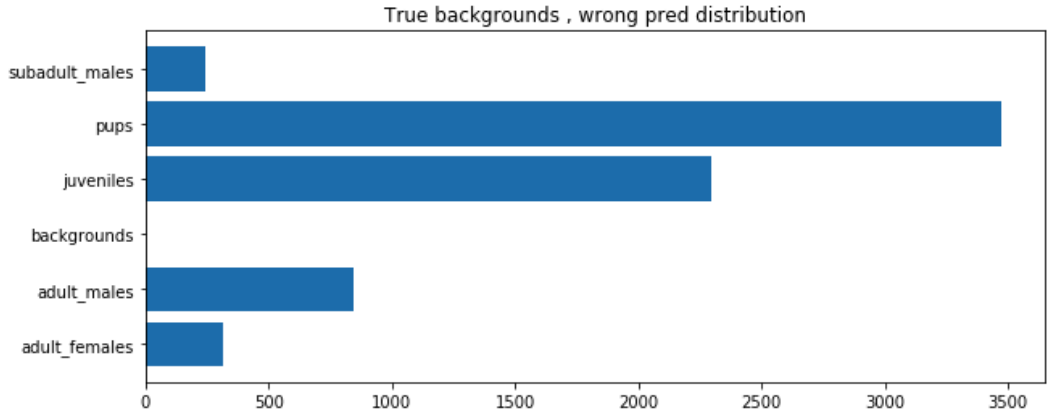
As we have observed before, adult_females most likely to be classified as juveniles because they share a similar size.

3.True adult_males



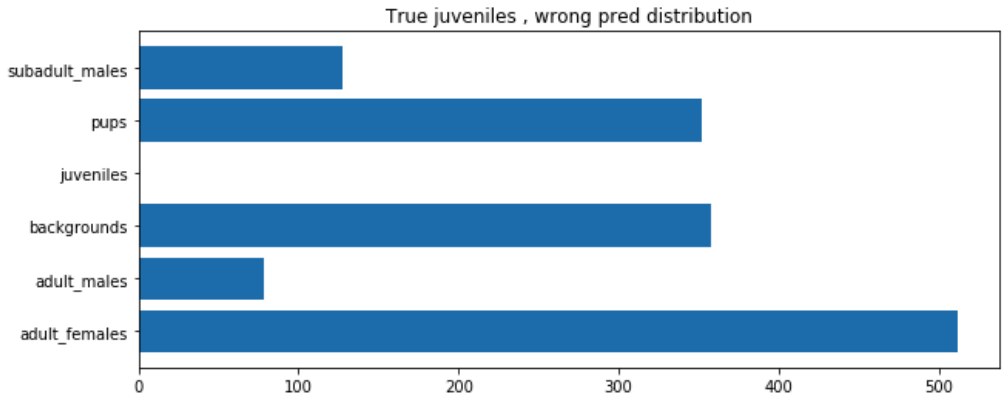
Adult males mostly likely to be classified as subadult_males. Also for the reason that they share a very similar size.

4.True backgrounds



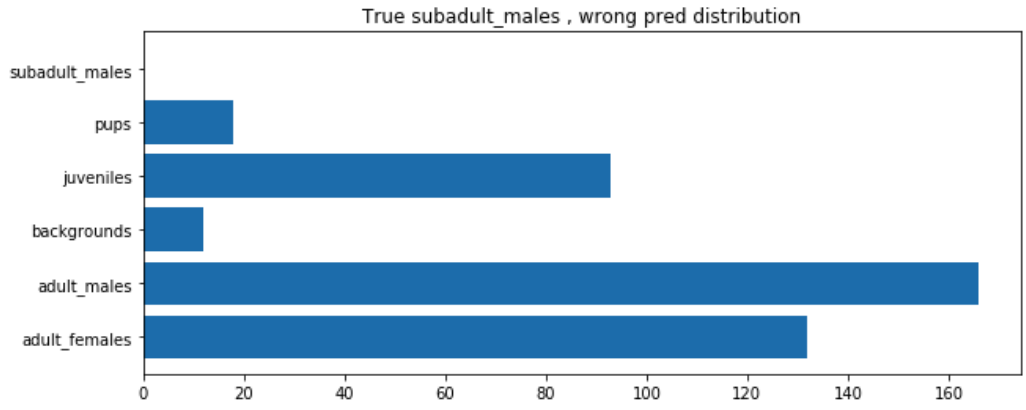
Background patches mostly likely to be classified as pups. This is consistent with our previous observations.

5.True juveniles



As we can see, juveniles are troublesome. Their size varies a lot and they always show up with other sea lions.

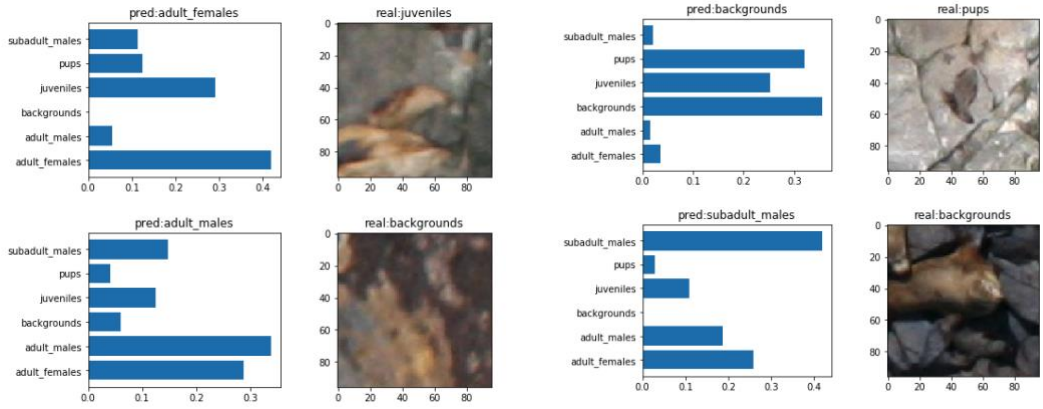
6.Subadult_males



Subadult males have similar sizes with adult males, so they are often misclassified as adult_males.

In a word, we see that the mistakes we made in the classification phase are reasonable. From our point of view, sea lions most likely to be distinguished by their size. When the size is not so different, the classification performance is not so good.

We randomly select some misclassified patches to illustrate:

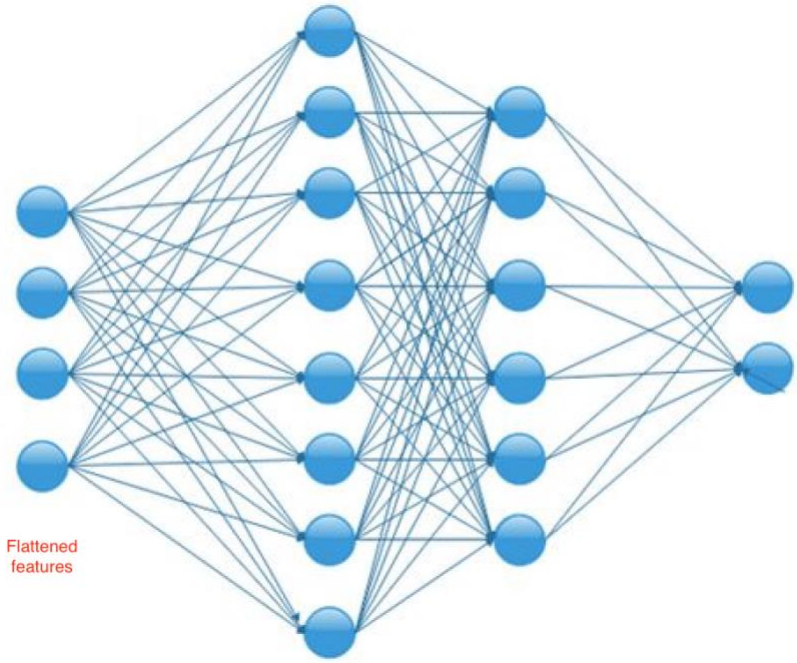


4.4 Fully Convolutional Network

In this project, we examine the idea of fully convolutional network. In CNN architecture, we usually have the following structures:

Image ---> [conv, pooling] * N ---> Flatten ---> Fully Connected Layers

Convolution and pooling operations can be seen as extracting features from images. Unlike logistic regression, we don't need to manually create features by ourselves. Convolution and pooling layers do that for us. The flatten layer takes the extracted features as input and makes it a vector. We can exploit these features using fully connected layers, just like the standard deep learning architecture. One disadvantage of fully connected layers is that we can not change the dimension of input. Suppose we have an image of size 96x96x3, after several convolution and pooling operations, we get a feature map of size 4x4x10, then the flattened feature vector will be 160x1. If we change the input image size, the output flattened feature vector will have a different size too, and we can not input this new feature vector into fully connected layers, since the weight dimension does not match.



Fully Convolutional Network is a CNN architecture without fully connected layers. The core idea is to transform the fully connected layers into convolutional layers. In fact, there is one to one mapping between convolution and fully connection. The details can be found in [2].

Let's now discuss about the advantage of converting fully connected layers into convolution layers:

1. No restriction about input dimension

Since fully connected layers are removed, there is no restrictions about input image size. We can input various sized images, and get different sized feature maps. So we don't need to crop images into a fixed size before feeding into CNN.

2. Batch wise training

As described in [2], fully convolution operation produces the same result as batch wise training. Training a fully convolutional network with an image is the same as training a fully connected CNN with batch of patches where the patches are extracted from that image. Although the result is the same, training in a fully convolutional way is a lot faster, since there are many overlapping computations avoided. Modern deep learning architecture optimizes convolution operation, and we can exploit it to speed up training process.

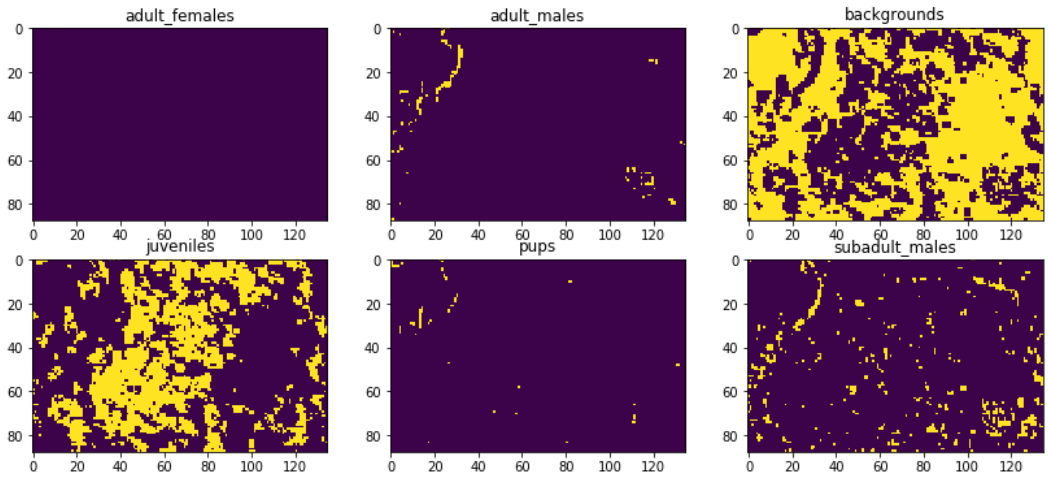
3. Fast inference

Recall that the image we have in this project is large, and we can not input it directly into

CNN, so we first extract patches from it. When we do inference on the test images, we need to first extract patches, classify each patch and then combine the results. This is quite time consuming. Now with fully convolution architecture, we can do it in a different way. After we get a trained model using patches, first convert the model into a fully convolutional network. Then when we do inference, we do not need to separate the image into patches, instead we can directly compute it.

We can also process the image in a fully convolutional way to examine the heat map. Here is an example:

We have totally six classes, the yellow color indicates that this pixel is classified as the corresponding class and the purple color indicates a negative answer.



As we can see, our model is quite good at classifying background patches. The heat map for the backgrounds has the shape of the environment in the original image. However, for sea lion patches, the result is not so satisfying. We can see that there are barely adult_females in this prediction but a lot of juveniles. This is because sea lions in this image are very close to each other and each patch could contain multiple sea lions. In the training dataset, juveniles tend to have this characteristic, so a lot of predictions are made as juveniles.

4.5 Transfer Learning

In order to improve the classification performance, we decide to use a more complex model. However due the the lack of training data and hardware limitations, training a deep CNN architecture is not feasible. So we tried transfer learning.

Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task. Transfer learning is popular in deep learning given the enormous resources required to train deep learning models. Modern deep learning architecture becomes deeper and deeper, and it contains millions of parameters to train. The training phase usually takes days, or even weeks. So it is a good idea to load a pre-trained model and start from there instead of totally from scratch.

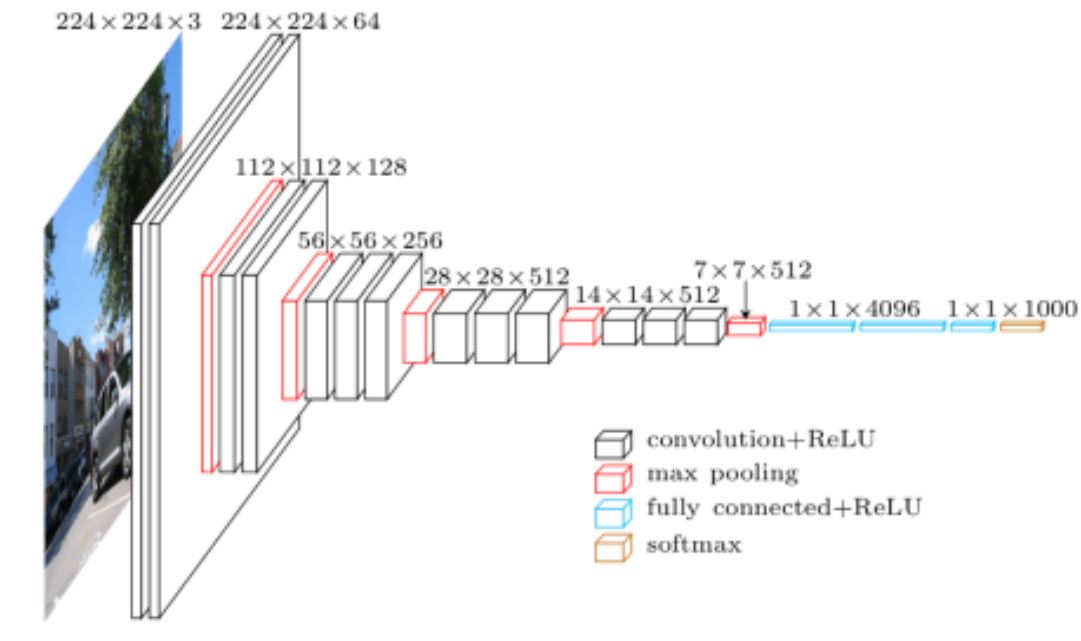
CNN architecture is very suitable to transfer learning, since convolutions and pooling operation abstract features of the images. These features are edges, colors, shapes which are often seen by different images. Thus we can re-use these weights for different tasks.

Transfer learning with a pre-trained model usually has the following procedures:

1. Modify fully connected layers to suit your problem
2. Freeze previous convolution and pooling layers and train only for fully connected layers
3. Make the whole net trainable with small learning rate

Some famous pre-trained models are VGG16, VGG19, GoogleLetNet, ResNet. We can load the pre-trained weights and do some fine tuning using our dataset. We experiment with VGG16 model.

VGG 16 model

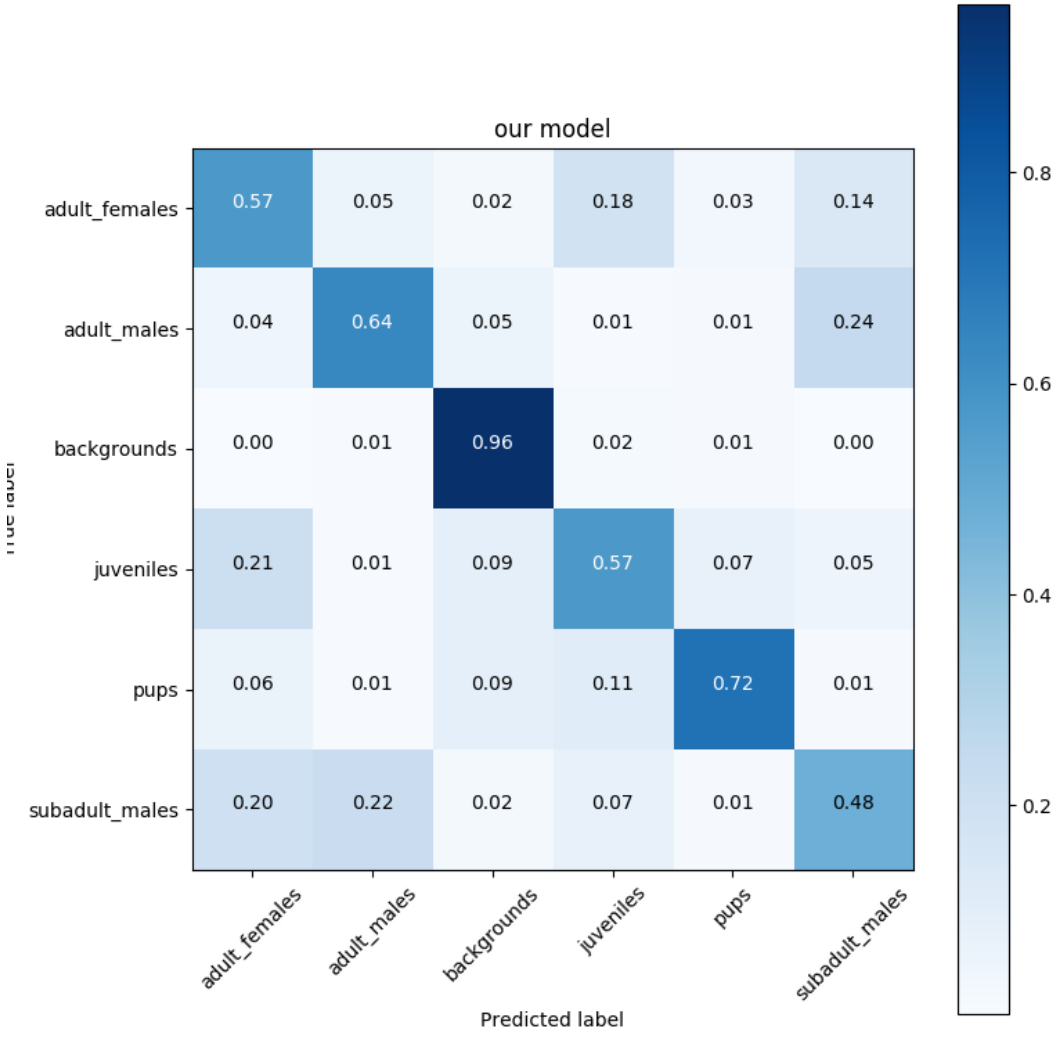


As we can see in the VGG16 architecture, after convolution and pooling operations, we have two fully-connected layers: one with 4096 hidden neurons and one with 1000 neurons softmax. The output has 1000 softmax units because the original VGG16 architecture deals with 1000-class classification problem.

In order to use this architecture in our problem, we first discard the fully connected layers and then add three fully connected layers: the first two are hidden layers with 256 neurons each and the last is a 6-class softmax layer. The whole training process is the following:

1. Discard original VGG16 fully connected layers and add our own
2. Freeze all the convolution layer weights and train the net to update the weights in fully connected layers for 20 epochs
3. Decrease the learning rate and make the whole network trainable. Train for 40 epochs.

Since the VGG16 architecture is more complex and deeper than our network, the computation takes more time, but the result is more satisfying. Here is the confusion matrix for VGG16:



Compared to our model, VGG16 achieves superior precision and recall. For each sea lion class, the true positive value is the highest.

5. Conclusion and Discussion

In this project, we start to build a CNN architecture from scratch using Keras framework. Using blob detection, we extract each sea lion patch with a sealion in the center. We also extract background patches as negative samples. During training, we tried different learning rates and different optimizers, for example SGD, SGD with momentum, AdaGrad, RmsProp, and Adam. They do not have significant difference for the performance score. In order to balance the dataset, we tried two approaches: assign different class weights and up/down sampling. All these approaches achieve better score than non-balanced dataset. We experimented with VGG 16 model and did some transfer learning. The VGG16 model achieves better score than our model, and this is reasonable because it is more

complex and deeper. However, training a model like this from scratch is not possible, because we do not have enough dataset and the training phase would take a long time.

Treating the sea lion counting task as a classification problem is one way to solve this Kaggle competition, we could also tackle this as a regression problem. Instead of classifying each patch, we could try to produce sea lion counts in each patch. We can remove the softmax layer and directly train the network with (patch, sea lion counts) data. In this project, we experiment with classification, regression can be an experiment in future.

6. Further Improvement

6.1 Counting problem

The tricky part for classifying each patch is to use a sliding window to extract patches. How should we do it properly? The naïve way is to do it in a non-overlapping manner. Different patches have no overlaps, and the final counting result is just a simple addition for all the patches. However, there is an issue here. Some training patches are like this:

They tend to appear close to each other. If we do the sliding window in the above way, we will get very in-precise result. The way to solve this is to do a smart sliding window with overlapping. But we need to figure out a way to normalize the result since each pixel is counted many times.

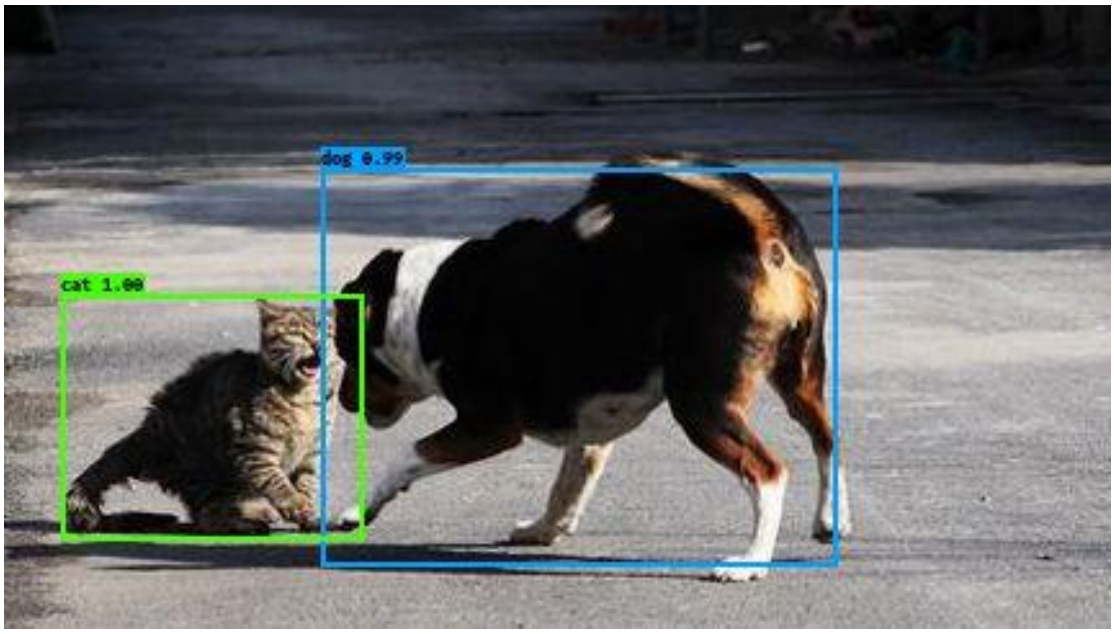
Another way to solve the counting problem is to do a regression network mentioned before.

6.2 Spatial Correlation

As we observed in during our project, there are some spatial correlation information that can be used. For example, pups seem to be close to their moms. Juveniles tend to stick together, Adult males seem to be far away from others. We can exploit this kind of spatial information to increase our model performance.

6.3 Object Detection and Semantic Segmentation

The champions of this Kaggle competition use the idea of object detection. They use the method of faster RCNN, UNet, Mask RCNN etc. Object detection is another hot topic in computer vision field where the algorithms are required to decide both the location and the type of objects in an image, for example:



Currently there are many existing solutions for object detection. The core idea is:

1. Generate ROI (region of interests which may contain an object)
2. Classify each ROI
3. Combine the results

Treating this problem as an object detection problem is more difficult than the direct classification approach. But this is an interesting idea and worth a try.

7. Bibliography

[1] Karpathy, A. (2016). *CS231n: Convolutional neural networks for visual recognition.*

[2] Bing S.(2016). *Improving Fully Convolution Network for Semantic Segmentation*

[3] Simard P.Y., Steinkraus D., and Platt J.C. *Best practices for convolutional neural networks applied to visual document analysis.* In Proceedings of the Seventh International Conference on Document Analysis and Recognition, volume 2, pages 958–962, 2003.

[4] Shuiwang J., Wei X., Ming Y., Kai Y., 3D Convolutional Neural Networks for Human Action Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 35 (1): 221–231 (2013)