Spring 2018

A Convolutional Neural Network Model for Species Classification of Camera Trap Images

Annie Casey
Boise State University, annie.casey15@gmail.com

A Convolutional Neural Network Model for Species Classification of Camera Trap Images

Boise State University

Annie Casey

Abstract

The overall purpose of this study was to automate the manual process of tagging species found in camera trap images using machine learning. The basic design of this study was to implement a Convolutional Neural Network model in Python using the Keras and Tensorflow modules that learn to recognize patterns in images in order to classify what species is in a given image and to label it accordingly. Results of the analysis highlight the importance of a large sample size, the degree of accuracy according to various arguments in the model, effectiveness of multiple layers that include Max Pooling, and limitations due to processing capacity. Findings include a summary of classification accuracy at varying predictive probability thresholds.

1 Introduction

Machine Learning

Machine learning implements computational algorithms for modeling large data sets. Machine learning models learn from given data and recognizes patterns to use with future data. Supervised machine learning involves using given labeled data as examples to predict future events. Such models "learn" by estimating statistical parameters using samples of labeled data, and comparing estimated output values against known values to adjust parameters that minimize error functions.

A general strategy for machine learning is the train-test procedure. A sample of data for which a response variable is known is randomly partitioned into a training subset and a testing subset - after fitting a model using data in the training set, the testing set can be fed into the model, with predicted outcomes compared to known outcomes in order to evaluate the performance of your model.

Mathematical Foundations of Image Recognition

In image recognition modeling, digital color images are encoded as lists of matrices, with each color channel (Red, Green, Blue) corresponding to a matrix of pixels. For example, the three matrices below might represent a 3 by 3 image of a purple "+" symbol.

$$[R, G, B] = \left[\begin{pmatrix} 0 & 255 & 0 \\ 255 & 255 & 255 \\ 0 & 255 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 255 & 0 \\ 255 & 255 & 255 \\ 0 & 255 & 0 \end{pmatrix} \right]$$

In a categorical classification problem for a large set of color images, a machine learning approach may involve using a train and test set of images to develop a probabilistic estimate of the likelihood that a particular list of matrices corresponds to each of the target categories.

Convolutional Neural Networks

Convolutional Neural Networks (CNN) mimic biological processes through analogy to the connectivity between neurons in the brain. Statistically weighted parameters in these models act similarly to when each neuron in the brain responds to stimuli in a certain way. CNN has been applied to subjects such as image and video recognition, recommender systems and natural language processing.

Convolutional Neural Networking for classifying species in an image involves training these neural networks on (ideally) many thousands of images of species and making the neural network learn to predict which class the image belongs to. Convolution parameters involve the use of filters and when multiple filters are used in sequence this creates greater recognition of smaller bits of the image. This means that the more filters, the better. Max pooling filters reduce the size of the representation. The combination of convolutional and max pooling filters creates thoroughly connected layers throughout the image.

Species Classification as an Image Recognition Problem

The Primate and Predator Project is based in the Soutpansberg Mountains of South Africa and is funded by Durham University's Anthropology Department. One side of the project focuses on "the

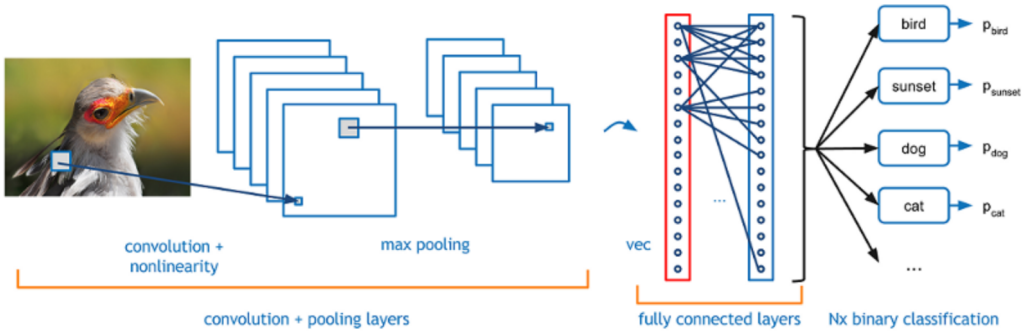

Figure 1: Image Recognition Using CNN

behavioral ecology of predator-prey interactions focusing on diurnal primates and their predators as a model system". The project uses camera traps to capture images of animal activity in the area. With permission from the project, we were allowed access to a portion of these images to use with our model. Although this is not a huge sample size to test a CNN model on, the small-scale attempt to construct a machine learning model has a great deal of potential in assisting The Primate and Predator Project in their manual tagging processes. Ideally, with a more advanced model, and more images to train, this model could help drastically reduce the need for manual species tagging of images, leaving the project with more time to focus on research and analysis.

Figure 2: Camera Trap Image

2 Methods

To run our desired model, we needed to import multiple Python modules:

- numpy and pandas - for working with structured arrays of data
- pil - for loading, manipulating, and saving images
- tensorflow - for computational implementation of neural network model layers
- keras - for semantic building and interpretation of multi-layer neural network models
- sklearn - for preparing and evaluating models in a machine learning context
- matplotlib - for plotting summaries of data

These packages allowed our model to use model classes such as Convolutional2D and MaxPool2D to specify a Convolutional Neural Networking model.

Images provided by the Primate and Predator Project were downloaded from a shared cloud storage folder and loaded as an array into Python. We resampled each image to a reduced size of 128 pixels x 128 pixels. This "thumbnail" size can be adjusted for higher or lower resolution modeling. The images were originally non-square, so in order to make each image an even 128x128 square we padded each image with zeros where needed prior to resaving.

In order to test if our model's prediction was correct we needed to create a new array of labels corresponding to the images. This was as easy as converting a spreadsheet with hand-coded species labels corresponding to each image into a Python array.

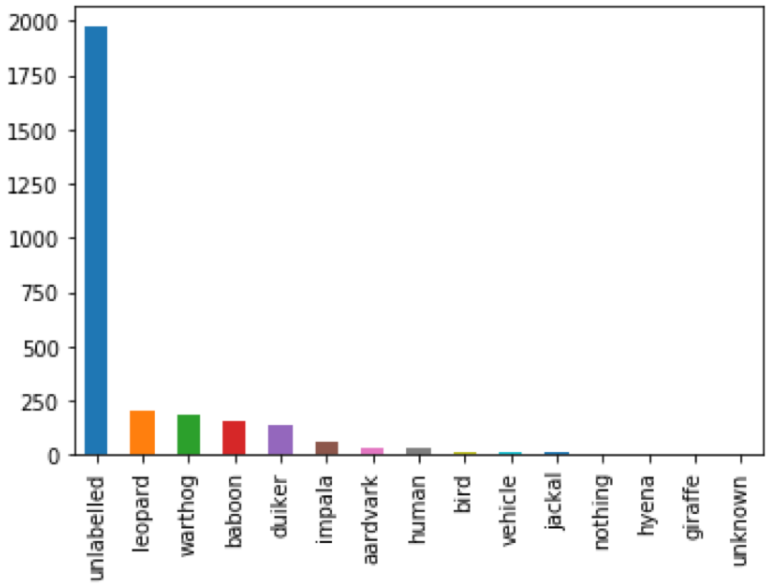

Figure 3: Distribution of all images

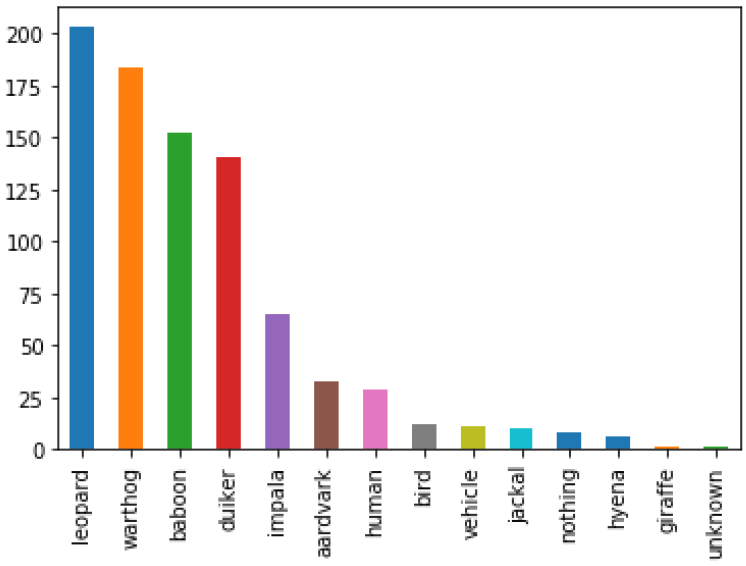

Figure 4: Distribution of labeled images

Due to small sample sizes across the less common species, only the top five most frequent species from the images were modeled, while all the rest were binned into an "other" category. Ultimately, the response variable in the model allowed for classification of each image into one of {Leopard, Warthog, Baboon, Duiker, Unlabeled, Other}.

A machine learning method called Data Augmentation helped us make the most of our small

sample size. Augmenting data involves applying transformations on existing data such as zooming in, zooming out, rotating and shifting up or down. This creates more data from our base data and thus a greater sample size to train. This process also helps in image classification by training the model to observe a leopard in a picture whether it's positioned differently, close up or far away, etc.

The next step was to split images into training and testing sets. We chose to randomly divide 60% of our images into the training set, withholding the other 40% of images for our testing set. This means our model was going to only use the images in the training set for building a numerical classification model. A larger training set can help build a more robust model while reducing the likelihood of over-fitting the model to the specific images in the training set.

The training and testing sets were each accompanied by the corresponding labels. We ended with four different variable arrays matching the testing set images, the testing set labels, the training set images and the training set labels.

Species	Count
Baboon	94
Duiker	81
Leopard	129
Other	107
Warthog	102

Table 1: Training Set Species Count

Species	Count
Baboon	58
Duiker	60
Leopard	73
Other	69
Warthog	82

Table 2: Testing Set Species Count

After dividing images, assigning labels, importing modules, and reshaping data, we started the model fitting procedure. The model included three convolutional layers with max pooling (Figure 6) added on top of each other and normalized. Each layer requires image pixels to be trimmed and the shape of the image to change. This also leads to non-trainable parameters.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 126, 126, 20)      560
_________________________________________________________________
batch_normalization_1 (Batch (None, 126, 126, 20)      80
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 63, 63, 20)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 61, 61, 50)        9050
_________________________________________________________________
batch_normalization_2 (Batch (None, 61, 61, 50)        200
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 30, 30, 50)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 26, 26, 50)        62550
_________________________________________________________________
batch_normalization_3 (Batch (None, 26, 26, 50)        200
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 13, 13, 50)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 8450)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               2163456
_________________________________________________________________
dense_2 (Dense)              (None, 5)                 1285
=================================================================
Total params: 2,237,381
Trainable params: 2,237,141
Non-trainable params: 240
```

Figure 5: Model Summary according to each layer added

Our model assigned predictions based on which of the five estimated species probabilities was largest. We set a threshold for how high we need the probability to be in order to classify an image. Any images for which all estimated species probabilities were below the threshold were classified as "unsure", due to the fact that we would prefer greater accuracy on a smaller number of images rather than a smaller degree of accuracy on a large number of images. To match the data

format needed for modeling, we hot-key encoded the categorical species labels, including "unsure", into binary representations of numerical values 0-5.

After compiling, our model compared its own predictions with the actual species label to formulate an overall accuracy level. We also used these predictions to create a confusion matrix displaying the true species (rows) with it's corresponding model prediction (columns).

Species	Numerical Value
Baboon	0
Duiker	1
Leopard	2
Other	3
Warthog	4
Unsure	5

Table 3: Species Assignment

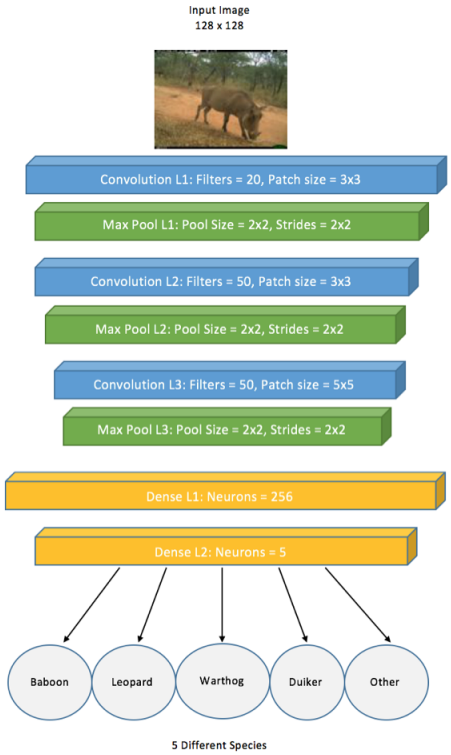

Figure 6: Convolutional Layers Per Image

3 Results

Altogether, the machine learning model used 2,237,381 total parameters and was able to train 2,237,141 of those total parameters, causing a total loss of 240 parameters. The arguments for the model-fit-generator included a number of epochs, steps per epoch, learning callback function, training and testing data, batch sizes, and more. We used 10 epochs and 50 steps per epoch to fit the model. Due to time constraints and computing capacity, we had to keep time-consuming arguments fairly small.

We set our model's probability threshold at 90%, this means that an estimated species probability for an image must be at least 90% in order for our model to classify the image.

Threshold	Proportion Predicted	Correct Prediction
0	100%	76.90%
0.75	95.46%	89.87%
0.80	94.81%	91.44%
0.90	92.26%	95.80%
0.99	87.07%	97.95%

Our final results returned a 64.05% loss and 76.90% accuracy.

- For Duiker images, 32 were labeled correctly, while 3 were labeled incorrectly and 28 did not have a large enough probability so they were labeled 'unsure'.
- For Leopard images, 53 were labeled correctly, while 0 were labeled incorrectly and 22 did not have a large enough probability so they were labeled 'unsure'.
- For Baboon images, 30 were labeled correctly, while 3 were labeled incorrectly and 26 did not have a large enough probability so they were labeled 'unsure'.
- For Warthog images, 21 were labeled correctly, while 2 were labeled incorrectly and 48 did not have a large enough probability so they were labeled 'unsure'.
- For Other images, 24 were labeled correctly, while 0 were labeled incorrectly and 51 did not have a large enough probability so they were labeled 'unsure'.

The plots of accuracy loss and overall accuracy suggest the model accuracy was beginning to plateau, although additional epochs could marginally improve model accuracy.

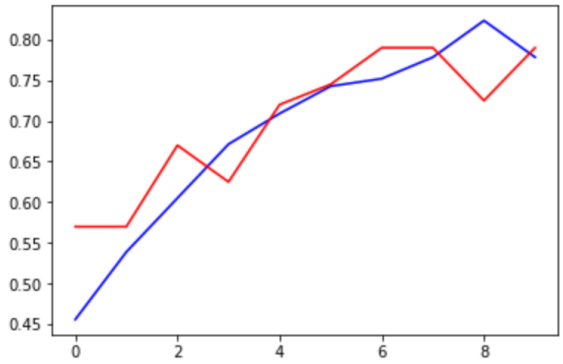

Figure 7: Final Accuracy Training vs Testing

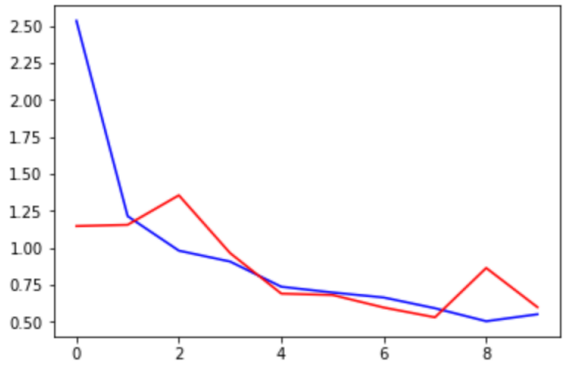

Figure 8: Final Loss Training vs Testing

4 Conclusion

With our results, we were able to cut down the amount of time spent on labeling obvious camera trap images and created an "unsure" label to provide a more narrowed pool of unlabeled images. Although our results included a majority of unsure labeled images, the images classified by the model were accurate.

There are many ways to improve this model. Adding more convolutional and max-pooling layers with greater filter and patch sizes is likely to increase the accuracy of the predictions. Increasing both number of epochs and epochs per step increases run-time but increases the number of repetitions from which the model can learn from. Instead of using 10 epochs at 50 steps per epoch, trying to using 20-100 total epochs at 100 steps each would also be appropriate.

Accumulating a greater sample size would increase the amount of data that can be used for training. The number of species being classified can also be adjusted according to the data set and sample sizes. Our model predicted four different species and 'Other', but with a greater sample size for each species, the model would be able to predict and train on a greater range of species. Adjusting the size of each image would also affect the prediction results. We assigned each image to be 128x128, but increasing these dimensions will increase image resolution and would provide potentially much more information to be fed into the model.

I learned that these modeling procedures are never truly finished and can be modified and improved as much or as little as needed. I am very passionate about this project because I have lived at the Primate and Predator Project research center in South Africa and have spent hours tagging these camera trap images. Even improving this tedious task by just a little bit would still lead to more time for research and analysis for the project. This model can now be the platform for future projects to use and improve.

5 Appendix

```
"""
loading and wrangling camera trap image data for CNN

"""

import os, sys, glob, re, numpy as np, pandas as pd
from __future__ import print_function
from keras.utils.np_utils import to_categorical # convert to one-hot-encoding
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D, BatchNormalization
from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import LearningRateScheduler
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
from PIL import Image
import shutil

#### CHANGE THIS FOR YOUR MACHINE
ac_path_pc = "/Users/control/Desktop/AnniesThesis"
os.chdir(ac_path_pc)
####################################

"""
PREP IMAGES FOR MODELING
"""
imgs = np.load("thumb_data_arrays.npz")
imgs = imgs['arr_0'] # image data is first file in archive


# pad or crop to same dimensions (square?)
nsquare = 128
# loop through the list of images, pad with 0s to make square, and resave the image
for ind in range(len(imgs)) :
    img = imgs[ind]
    if img.shape[0] < nsquare :
        imgs[ind] = np.pad(img,
                           pad_width = ((0, nsquare - img.shape[0]), (0,0),(0,0)),
                           mode='constant', constant_values=0 )



# reshape imgs from a one-dimensional array (of 3 dimensional arrays) to a 4 dimensional array
imgs_new = np.zeros((len(imgs), nsquare, nsquare, 3))
for i in range(len(imgs)):
    imgs_new[i] = imgs[i]

imgs = imgs_new

img_data = pd.read_csv('img_data.csv', sep=',')


# counts of images
```

```
img_data['label'].value_counts()

img_data['label'].value_counts().plot(kind="bar")

# collapse image labels for improved modeling results
img_labels = img_data['label']

labels_keep_n = 5

labels_sorted = img_labels.value_counts().index

# relabel all but the n most common labels as 'other'
for lab in labels_sorted[labels_keep_n:len(labels_sorted)]:
    img_labels[img_labels == lab] = 'other'


"""
Split images into train and testing sets
"""
# randomly mark a fixed percentage of "labelled" images as "train", the remaining as "test" (60-40

p_train = .6

def split_train_test(prop, indices):
    ntrain = round(prop*len(indices))
    np.random.shuffle(indices) # scramble the list of indices
    istrain = indices[0:(ntrain -1)] # keep the first set for training
    istest = indices[ntrain:len(indices)] # use the remaining set for testing
    return {'train' : istrain, 'test' : istest }

islab = np.array(range(len(img_data)))[img_data.label != 'unlabelled']
ind_split = split_train_test(p_train, islab) # contains imgs_split['train'] and imgs_split['test']

x_train = imgs[ind_split['train']]
x_val = imgs[ind_split['test']] # test is the same as "val" in our CNN tutorial

y_train = img_data.label[ind_split['train']]

# y_train = y_train.values.reshape(-1,1)

y_val = img_data.label[ind_split['test']]

# compare labeled values in the training and test sets
y_train.value_counts()
y_val.value_counts()

# convert pixel values to the interval 0 to 1
x_train = x_train.astype("float32")/255.
x_val = x_val.astype("float32")/255.


label_encoder = LabelEncoder()
label_encoder.fit(np.concatenate((y_train, y_val),axis=0))
y_train = label_encoder.transform(y_train)
y_val = label_encoder.transform(y_val)

animals = label_encoder.classes_
```

```
# hot key encoding of categorical labels (the position of "1" in a vector of 0s indicates which la
y_train = to_categorical(y_train)
y_val = to_categorical(y_val)

# convert y arrays to lists (to match data format needed for modeling)
for i in range(len(y_train)):
    y_train[i] = y_train[i].tolist()

for i in range(len(y_val)):
    y_val[i] = y_val[i].tolist()

"""

START MODELING

Following the Digit Recognizer tutorial for CNN:
https://www.kaggle.com/toregil/welcome-to-deep-learning-cnn-99

"""

""" Specify Model """
model = Sequential()

#first set of CONV > POOL Layers
model.add(Conv2D(filters = 20, kernel_size = (3, 3), activation='relu',
                 input_shape = (128, 128, 3)))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size = (2,2), strides=(2,2)))

#second set of CONV > POOL Layers
model.add(Conv2D(filters = 50, kernel_size = (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size = (2,2), strides=(2,2)))

#third set of CONV > POOL Layers
model.add(Conv2D(filters = 50, kernel_size = (5, 5), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size = (2,2), strides=(2,2)))


# first (and only) set of FC => RELU layers (REctified Linear Units)
# https://github.com/Kulbear/deep-learning-nano-foundation/wiki/ReLU-and-Softmax-Activation-Functi
model.add(Flatten())
model.add(Dense(256, activation='relu'))

#softmax classifier
model.add(Dense(labels_keep_n, activation='softmax'))

""" Inspect Model (fine-tune specification based on numbers of parameters) """
# view numbers of parameters and dimensions of outputs in model
model.summary()


""" Compile and run model """
datagen = ImageDataGenerator(zoom_range = 0.1,
                             height_shift_range = 0.1,
                             width_shift_range = 0.1,
                             rotation_range = 10)
```

```
model.compile(loss='categorical_crossentropy', optimizer = Adam(lr=1e-4), metrics=["accuracy"])

annealer = LearningRateScheduler(lambda x: 1e-3 * 0.9 ** x)



hist = model.fit_generator(datagen.flow(x_train, y_train, batch_size=16),
                           steps_per_epoch=50, # try adjusting to improve accuracy without taking
                           epochs=10, # increase this to 20-100 when running the final model
                           verbose=2,  #1 for ETA, 0 for silent
                           validation_data=(x_val[:200,:], y_val[:200,:]), #For speed
                           callbacks=[annealer])


"""
Evaluate Modeling Results

"""

final_loss, final_acc = model.evaluate(x_val, y_val, verbose=0)
print("Final loss: {0:.4f}, final accuracy: {1:.4f}".format(final_loss, final_acc))

plt.plot(hist.history['loss'], color='b')
plt.plot(hist.history['val_loss'], color='r')
plt.show()
plt.plot(hist.history['acc'], color='b')
plt.plot(hist.history['val_acc'], color='r')
plt.show()

x_val_unlab = imgs[img_data.label == 'unlabelled']


x_val_all = np.append(x_val, x_val_unlab, axis=0)

y_hat = model.predict(x_val_all)

animals = np.append (animals, ["unsure"])


pred_thresholds = [0, .75, .8, .9, .99]  # set to 0 to predict on all the testing images
for pred_threshold in pred_thresholds:
    print("******************************")
    print("******** SHOWING RESULTS FOR ***")
    print("***** Prediction Threshold = "+str(pred_threshold))
    y_pred = np.argmax(y_hat, axis=1)

    for i in range(len(y_pred)):
        y_hat_maxprob = y_hat[i].max()
        if y_hat_maxprob < pred_threshold:
            y_pred[i] = labels_keep_n


    y_true = np.argmax(y_val, axis=1)
    y_true = np.append(y_true, np.repeat(labels_keep_n, len(x_val_unlab)))

    cm = confusion_matrix(y_true, y_pred)
```

```
    cm_correct = sum(cm[i][i] for i in range(len(animals) -1 ))
    no_guess = np.count_nonzero( y_pred == labels_keep_n)

    # number we classified is the "interior" of the matrix, after removing last row and column
    did_guess = len(y_hat) - no_guess - len(x_val_unlab) + cm[labels_keep_n][labels_keep_n]

    print(cm)
    print(animals)

    print("How many images were classified?")
    print( did_guess)

    print("How many images were classified correctly?")
    print(cm_correct)

    print("For what proportion of images did we make a prediction?")
    print ( 1 - no_guess / len(y_hat) )

    print("For what proportion of our predictions was the prediction correct?")
    print( cm_correct/ did_guess)

    """
    Show Images Labeled by Model
    """

    out_dir = "test_img_guesses"+re.sub('\\.','_', str(pred_threshold))
    if(os.path.exists(out_dir)):
        shutil.rmtree(out_dir, ignore_errors=True)

    for i in range(len(np.unique(y_pred))):
        # print (animals[i])
        subdir = os.path.join(out_dir, animals[i])
        ggdir = os.path.join(subdir, "good_guess")
        bgdir = os.path.join(subdir, "bad_guess")


        os.makedirs(ggdir, exist_ok=True)
        os.makedirs(bgdir, exist_ok=True)
        for j in range(len(y_pred)):
            if y_pred[j] == i:
                im = x_val_all[j]
                #Rescale to 0-255 and convert to uint8
                im = (255.0 / im.max() * (im - im.min())).astype(np.uint8)

                im = Image.fromarray(im)
                if animals[y_true[j]] != "unsure":
                    if(y_pred[j] == y_true[j]):
                        # good guess
                        outfile = os.path.join(ggdir, animals[i]+str(j) )
                    else:
                        # bad guess
                        outfile = os.path.join(bgdir, animals[y_true[j]]+str(j))
                    im.save(outfile+".jpeg")

"""
manually inspect y_hat (probabilities of each species) for one or more images
"""
y_hat[305:309]
```