Capstone Project 2 Milestone Report

Classifying Images of Spotted Cat Breeds

Problem Statement

Suppose you have a certain passion or curiosity for large felines and you want to learn more about them. From all the way back to my middle school days, students would be placed in different groups named after cat breeds. After some time, I have seen various pictures of such creatures and thought, “Leopards, cheetahs, and jaguars look about the same. How are you supposed to distinguish between them?”

Each cat breed has different attributes which help us distinguish between one other. Lions (males especially) are known to have thick fur around the front area; tigers have orange skin with black stripes; black panthers are obviously recognizable for their dark skin. However, you find yourself scratching your head whenever you encounter a spotted creature: very often, you confuse yourself over whether it is a cheetah, leopard, or jaguar. At that point, you wish there is something that can help clear your doubts.

Potential Audience

The target audience for this project could be diverse, but for this project, it aims at:

- Feline fanatics
- Feline researchers
- Image recognition application developers

Data

For this project, since it aims to produce a image classification model, the dataset will consist of images belonging to one of several categories, namely:

- Cheetah
- Jaguar
- Leopard
- Snow leopard

The first challenge I encountered was retrieving the dataset. Researching for ways to obtain images from online, I tried the GoogleImageDownloader module, only to find out it is no longer practical. Instead, I stuck with an extension from Google

Chrome that scrapes images off a website. Using the tool on Google and Bing, I found them to be rather unreliable as they provided some unhelpful or irrelevant images, even with advanced searching. Fortunately, I found a image search site called **alamy.com**, which can reliably provide relevant images.

To get optimal results, I searched using their scientific names given by the binomial nomenclature. With each page displaying 100 images, I used the extension to download them until I reached 1000 images for each category. Then, I repeated the process for the other categories.

The main challenge of wrangling data begins here. Despite the site's reliable sources, the extension itself was not perfect. Aside from the relevant images, it also contained images of graphics relevant to the website's core graphics. Moreover, I had to extract all images from each subdirectory for each category. To do this, I ran some shell commands similar to below:

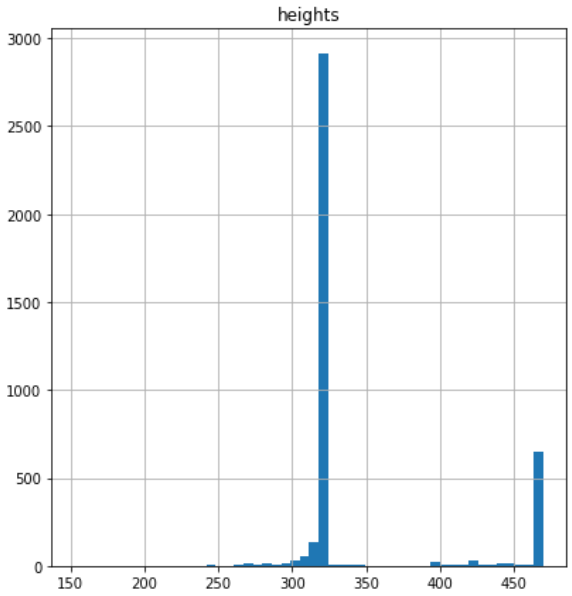
```
# This moves all images out of the subdirectories to their main
categorical directories
for d in */;
    do mv "$d"/*/* "$d";
done

# remove all empty directories
find . -type d -empty -delete
```

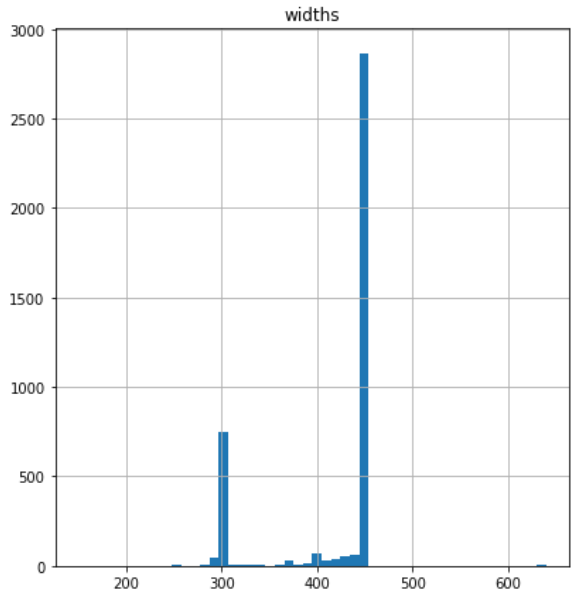
Next, I filtered out the aforementioned irrelevant images, which fortunately all either had a different file format or were of small size. However, due to the extensive file size range of images, I ended up deleting some of the images. After that, I obtained some more images so that each dataset category will contain 1000 images, totalling 4000.

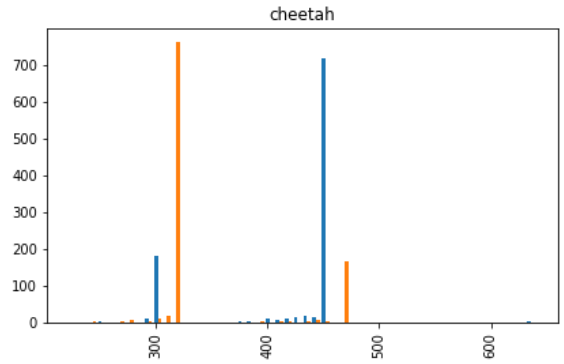
Some dataset analysis

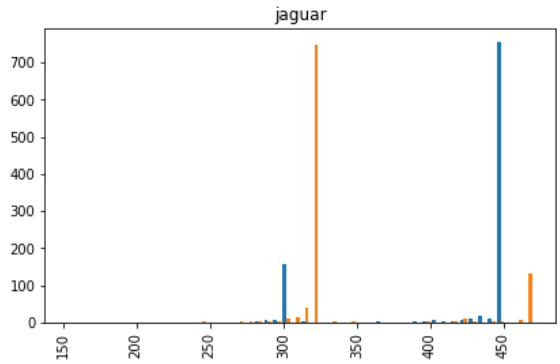
Due to the unstructured nature of an image dataset, it is impractical to provide statistical analyses.

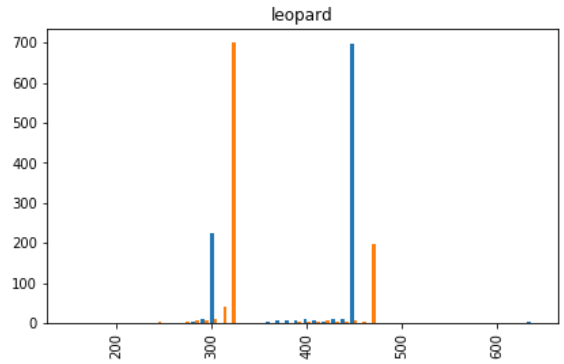
Here are some visual analyses for the image dataset. This would help me determine the sizes in which to resize each image in the dataset into the model.

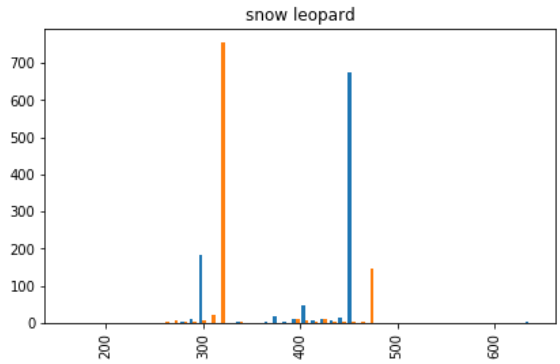








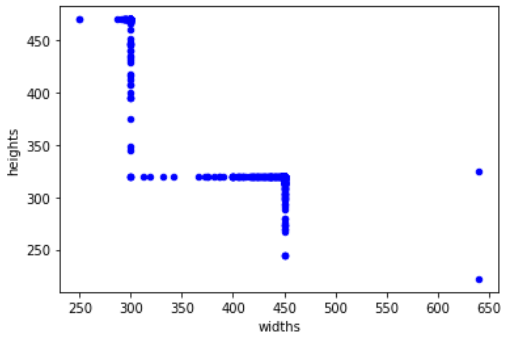


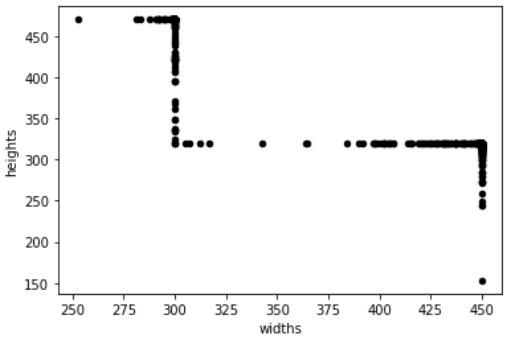


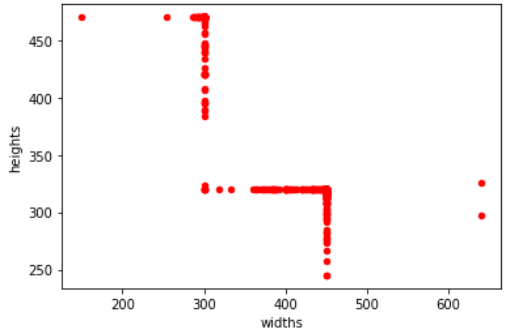
Here are the scatter plots visualizing the images' sizes, four for each individual category, and one for the whole dataset next page.

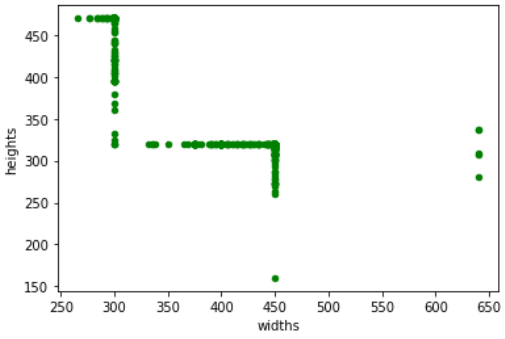
Color keys:

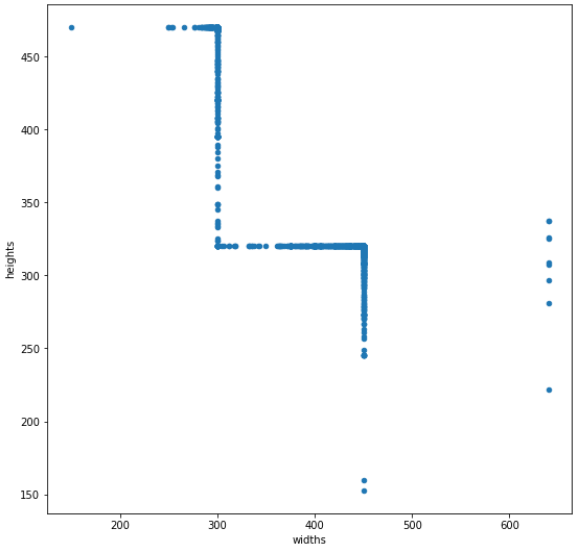
- Cheetah - **blue**
- Jaguar - **black**
- Leopard - **red**
- Snow Leopard - **green**











Looking at the data visualizations of the images' dimensions, I found that the images clustered most around the points (300, 450) and (450, 320). However, from the

former graphs, the most common widths and heights were 450 and 320, respectively. Thus, I have decided to use the size (160, 160) for resizing.

Image Classifier Model

For this project, there are several libraries that allow you to build a image classifier model such as Keras, PyTorch, and FastAI. Ultimately, for its simple yet effective use, I chose Keras.

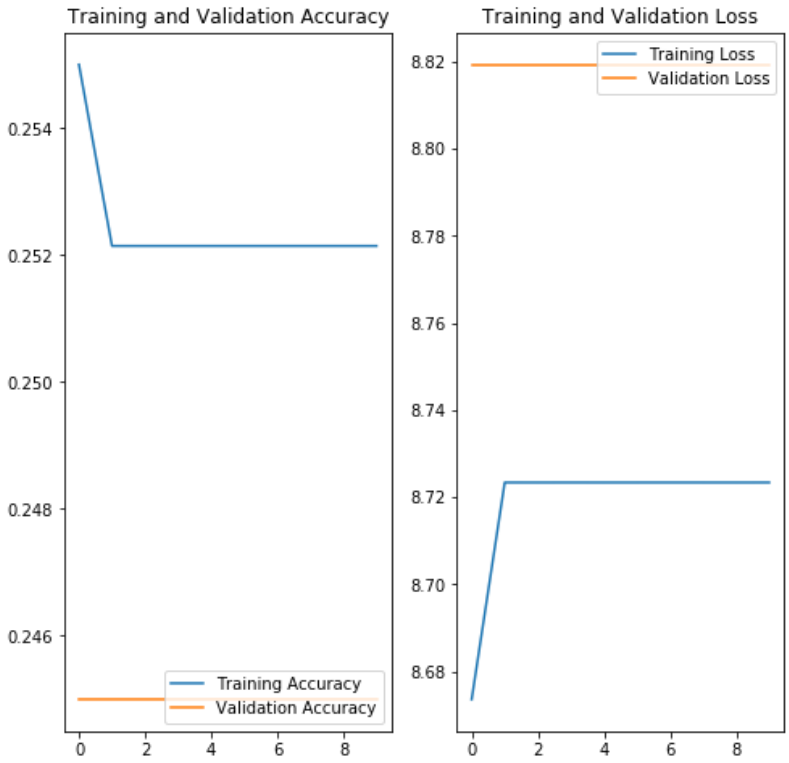
Simple Model

The first model I used used only a few layers, consisting of a preprocessing Rescaling layer, one Convolutional, a MaxPooling2D, a Dropout, which is supposed to help prevent overfitting, a Flatten, then two Dense layers in that order. To visualize:

Input: Image (160x160, 3 color channels)
↓
Rescaling: 1/255 for color channels, (160, 160, 3)
↓
Conv2D: 'relu' as activation function
↓
MaxPooling2D
↓
Dropout: 0.2 (to help prevent overfitting)
↓
Flatten: (160, 160) to array of length 160*160
↓
Dense: 64
↓
Dense: 4 (number of classes)

The preprocessing layer in this model only resized to the target size without any other color alterations. The last Dense layer is where the model outputs its prediction of one of the four categories.

Using the Adam optimizer and `sparse_categorical_crossentropy` as the loss function, after 10 epoch, the model's validity was evaluated:



The model achieved training accuracy of 24.3% and a validation accuracy of 24.5%.

Model with More Convolutional Layers

Due to the model's simple architecture, it did not perform well. In an attempt to rectify this, I added a data augmentation layer after the rescaler, which would alter each image in various ways such as flipping horizontally, zooming, and rotating.

Next, I added a couple more Conv2D layers, with MaxPooling2D layers between each Conv2D layer. The last MaxPooling2D then would then lead to the final layers, which would narrow down to their predictions.

Even then, the new model only attained a training accuracy of 24.7% and 25.5% validation accuracy after 10 epochs.

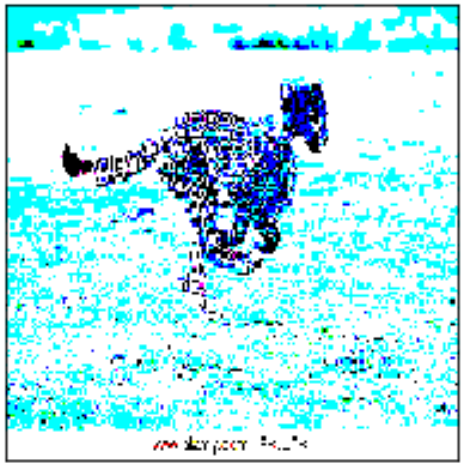
Transfer learning with ResNet

The two previous models seemed too simple to classify images with high accuracy. Thus, I finally decided to apply transfer learning, where the base model consists of layers with a portion of them frozen.

Since there are various transfer learning models, for simplicity, I decided to use ResNet50, which served as the base model. Following the base model were Flatten, then four Dense layers:

Input Layer
↓
ResNet50 (frozen)
↓
Flatten
↓
Dense (64)
↓
Dense (32)
↓
Dense (16)
↓
Dense (4)

In addition, I made drastic changes to the preprocessing layer. Instead of a validation split of 0.3, I used 0.2. Also, ResNet50's preprocessor not only resizes the input image, but also alters its colors. For example:

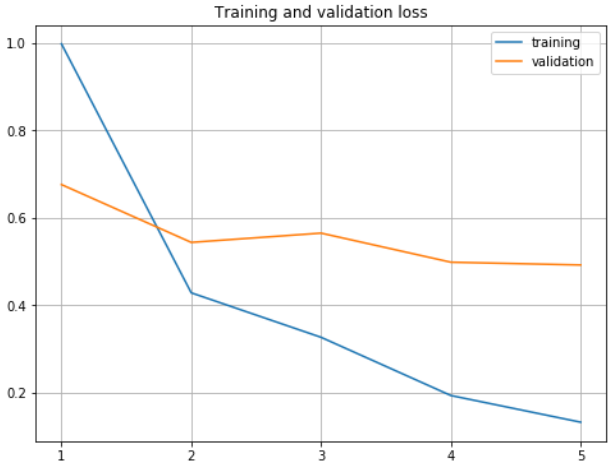


The first two models used images like the left with no color alterations, while the ResNet50 used images like the right rendered by its preprocessing function.

I compiled the ResNet model:

```
model.compile(loss='categorical_crossentropy',
                  optimizer=keras.optimizers.Adamax(lr=0.001),
                  metrics=['accuracy'])
```

After 5 epochs with 50 steps each:



The model attained a much better performance with 96.0% training accuracy and 84.6% validation accuracy.

Fine-tuning parameters

The model is vastly improved with using the ResNet50 architecture, but there are still ways to improve it even further. However, due to time constraints, for simplicity I have only made marginal changes to the model, namely its **learning rate** and target image size. A couple other things I considered were the number of epochs and dataset for the model generated from Keras's `ImageDataGenerator`.

List of changes tried:

- Add rescaling function, `rescaler=1/255` that alters colors for each image, into `ImageDataGenerator`
- Set learning rate lr = 0.01 instead of 0.001
- Set target size `IMAGE_SIZE = (160, 160) -> (200, 200)`
- Set number of epochs: 5 -> 8, 10

Results:

	Img Size	Rescaler	Learning Rate (lr)	# Epochs	Train Acc	Valid Acc	Train Loss	Valid Loss
Original	*160x160*	*None*	*0.001*	*5*	*96.04%*	*84.62%*	*0.1417*	*0.4916*
Rescale	160x160	**1./255.**	0.001	5	43.12%	42.88%	1.2790	1.2671
Change size	**200x200**	None	0.001	5	97.72%	87.00%	0.0915	0.4196
Change lr	200x200	None	**0.01**	5	96.94%	85.37%	0.1311	0.8275

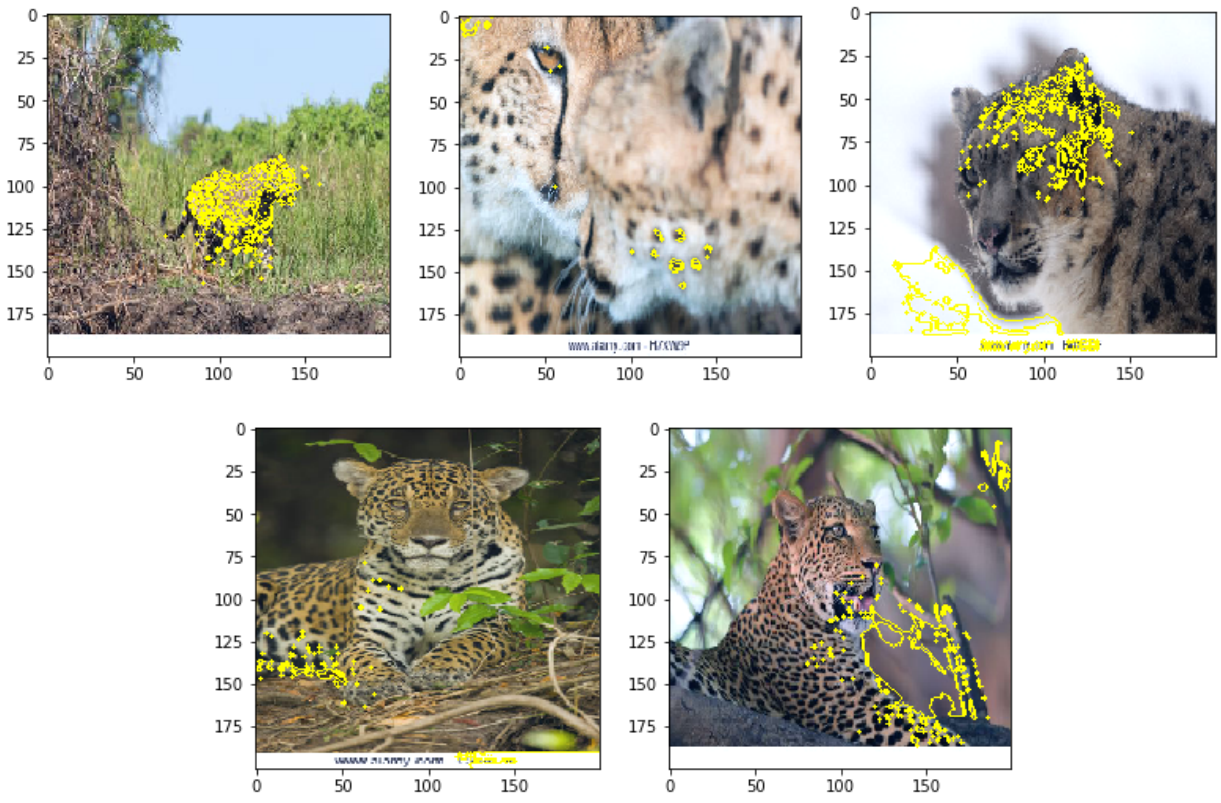
More epochs	**200x200**	**None**	**0.001**	**8**	**97.67%**	**88.25%**	**0.0871**	**0.5438**

Given the time constraints, fine-tuning hyperparameters thoroughly would be quite unfeasible, so I decided to use the model that performed the best. In this case, the model with increased size and more epochs had the best performance overall.

Interpreting Our Model

It is crucial that we know how and why the model predicts images the way it does. Pitting the numerical metrics against their corresponding images can help so much with diagnosis and whether we need to use other means to train the model. However, we can make the model analysis even more visual by tracing lines referencing the model's interpretation. The module to use is called LIME.

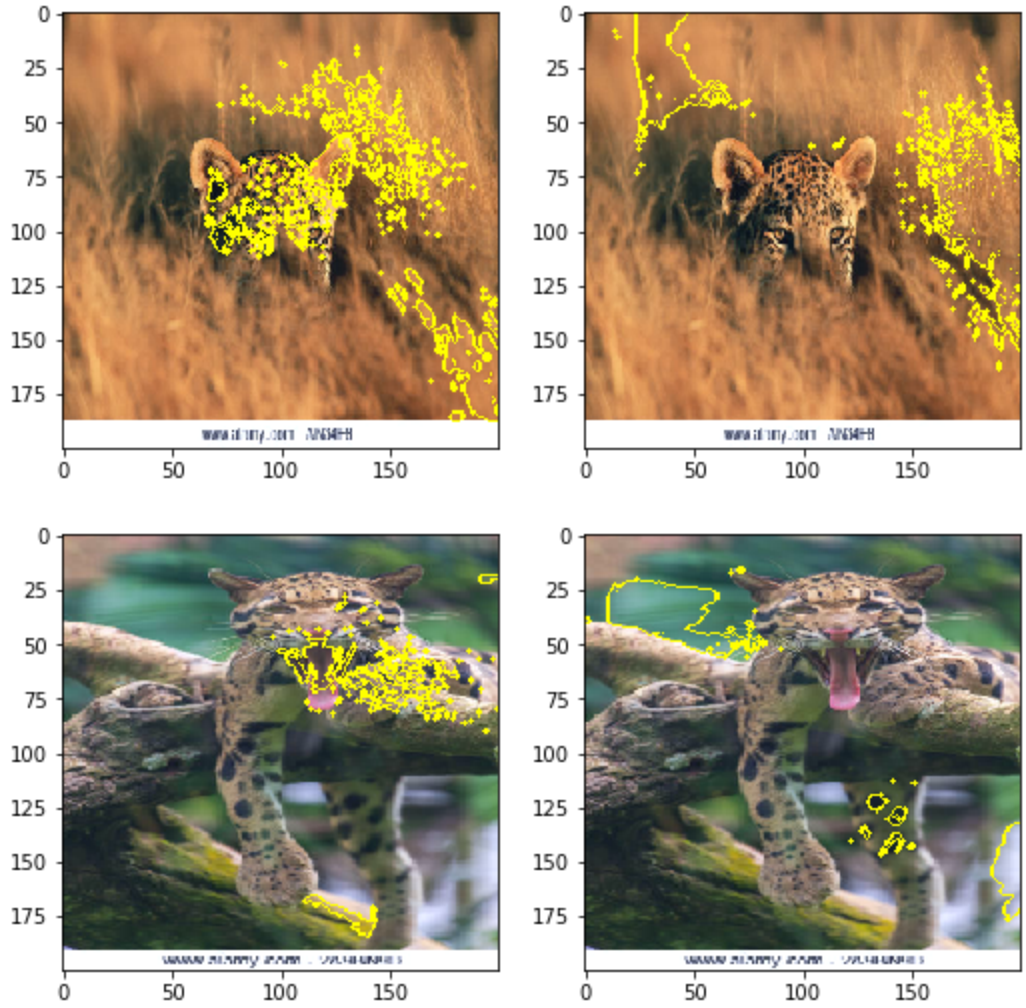
For best results, the module will examine the best model according to above.



The images above were some that were predicted with "100% (or close) accuracy." Though LIME's explanations are not perfect, it seems that it highlights the areas where the spots are, just as I desired, especially for the first and second images shown. The most interesting one is the third one, where it was correctly predicted as

“snow leopard,” in which LIME highlighted the white areas in the background and some of the spots on the creature’s head. Coincidentally, snow leopards are known to be spotted with white skin.

Below are a couple of the images that were predicted wrong.



For both images, the correct answers, leopard and snow leopard, were their second guesses, while their predictions were respectively cheetah and jaguar. However, it seems that both images did not show their bodies, their main distinguishing factors, which led to their mispredictions.

Limitations

Building a deep learning model (in a general sense) that is able to identify a category given an image would require a lot of dedication and time. Given the time constraint, we can only rely on creating simple models with a few convolutional layers or using

builtin architectures such as ResNet50 or ResNet101 which already have a complex system full of many convolutional layers. Otherwise, we would have spent the time constantly modifying the model such as modifying every layer's activation functions, trainability, and parameters.

Conclusion

Deep learning and its applications have come a long way in creating machines with increasingly more human-like intelligence. Yet, as with other types of deep learning applications, even with all the time we have in the world, it would take us unfathomable dedication if we were to build a model that can perfectly identify an object given an image, let alone a spotted cat breed. But to do so would also lead to the conclusion that humans (and animals) are just "highly advanced machines" built within organic matter.

But for now, I can only hope that this project is able to clear one's doubts toward identifying spotted cat breeds with applied deep learning.