Edwin He

31 Oct 2017 • on python keras machine learning CNN transfer learning

Machine Learning Engineer Nanodegree

Capstone Project

Edwin He
November 7th, 2017

I. Definition

Project Overview

Deep learning with Deep Neural Network has gained huge momentum in the past few years owning to the blooming of data and GPU accelerated computation. The former makes it possible to collect massive amount of training data to tune millions or even billions of parameters; the later makes running deep learning computation approachable for everyone have access to a consumer PC with GPU installed.

One of the most exciting fields of deep learning application is computer vision which gives machine the ability to see. One of the most influential innovations in the field of computer vision is Convolutional Neural Network(CNN), first published by LeCun et al [1], became prominence as Alex Krizhevsky used it to win 2012 ImageNet competition.

In this study, I will train an image classifier using the KTH-ANIMALS dataset[4] to classify 19 different animals with convolution neural network(CNN). Also look to utilize the trained weight for object localization.

Problem Statement

Though empowered by GPU, training an image classifier with CNN still requires a long computation time. CNN needs to go deeper to achieve good result on more complex tasks which requires massive amount of training data for tuning the weights. Neither is often feasible for individuals.

If we imagine how we learn as a human, we do not learn to recognize every object from stretch. After learning how a bike looks like and how it works, even for the first time we see a car, we can immediate transfer our knowledge to bikes and we can tell immediately a car is most likely a bike with more wheels and bigger in size.

Similarly after training a network to learn dogs and cats, we should not have to learn to classify a wolf and tiger from stretch because we have already learned how the face and body look like for dog or cat like animals.

There are many CNN models already trained with ImageNet (a massive image data set), i.e. it has learned to classify the objects in the ImageNet and have also learned the parts that assembles them. I will transfer the knowledge learned from classifying ImageNet objects to our tasks without having to learn from stretch. This will solve our problem as it greatly reduces the number of trainable weights in our model which requires way less data and effort to learn.

Metrics

Classification Accuracy, Cohen's kappa and Cross-Entropy Loss are used as evaluation metrics.

- Classification Accuracy is simply the percentage of correctly classified images, as the dataset is slightly imbalanced. This metric will be for reference only.
- Cohen's kappa is a score that expresses the level of agreement between two annotators, ranging from -1 to 1 with 1 being perfectly agreement. It is a more accurate performance measurement for model with imbalanced dataset. Cohen's kappa is defined by:

$$\kappa = (p_o - p_e)/(1 - p_e)$$

- Cross-Entropy Loss is a metric used with softmax function which generates probabilities for all outcomes. It will give me more information about how confident the model is for correct predictions as well as how far from correct the incorrect predictions are. Cross-Entropy Loss is also used as a metric to learn whether the model is improving or not for early stopping and model checkpoints. It is defined by[6]:

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{i,j}\log(p_{i,j})$$

II. Analysis

Data Exploration

The KTH-ANIMALS dataset[3] contains only 1740 images in total and are of different shapes from ~200 to ~400 pixels width and height. Figure 1 shows the images in their original shape.

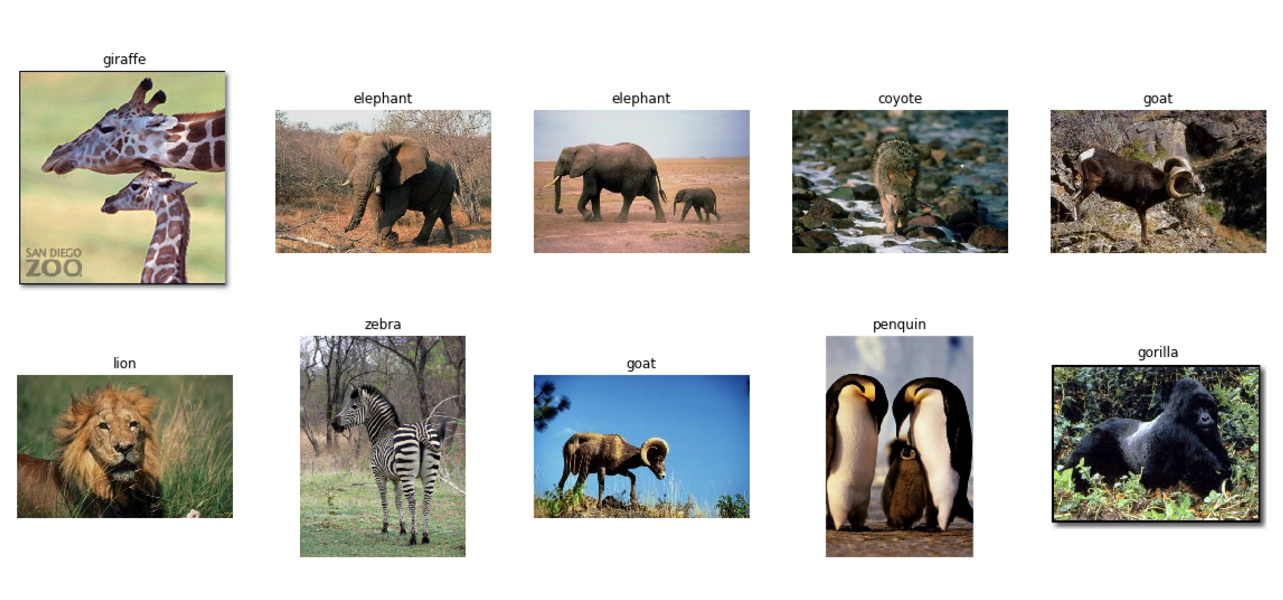


Figure 1: Images loaded as 224x224

Exploratory Visualization

Figure 2 shows the KDE plot of R,G,B channels for each class. It shows the color distribution for different class. I can see that animals in dark yellow such as coyote, horse, leopard, lion and tiger contributes to the peaks on the left of the red channel. Also in general, these pictures are taken in the wild, which explains the significant amount of dark green and dark blue in every most classes coming from the grass field and the sky.

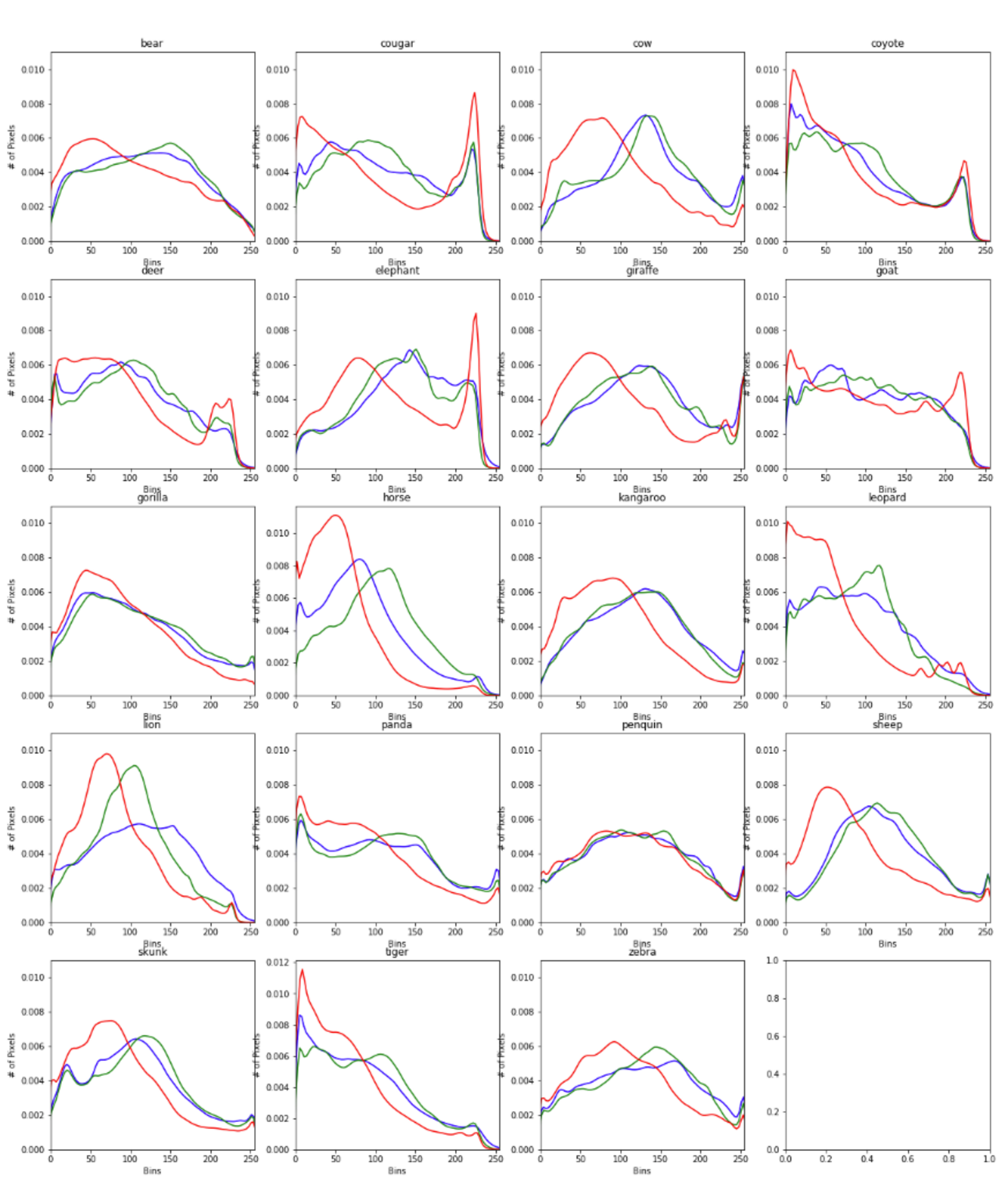


Figure 2: KDE plot by class

Figure 3 shows the label count. The dataset is quite evenly distributed.

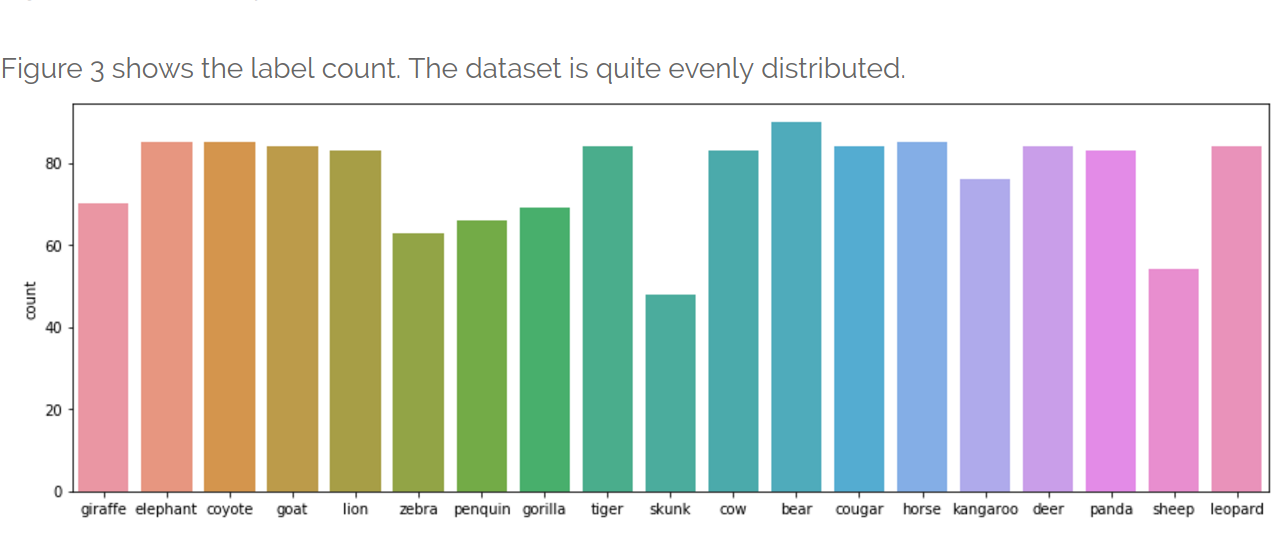


Figure 3: Training Lable Count per Class

Algorithms and Techniques

I use Convolutional Neural Network(CNN) to achieve this image classification task. CNN typically includes a few blocks of layers - each includes one of more Convolutional layers followed by Activation layer, Pooling layer and Dropout layer. Finally connecting to one or more fully connected(FC) layers for class prediction.

- Convolutional layer use n-by-n matrics called filters or kernels to slide through the input image performing matrix multiplication. The output becomes the input for the next layer. Figure 3 shows how the output is calculated from a 3x4 input, slided through by a 2x2 kernel with stride 1 (move 1 pixel right/down at a time).

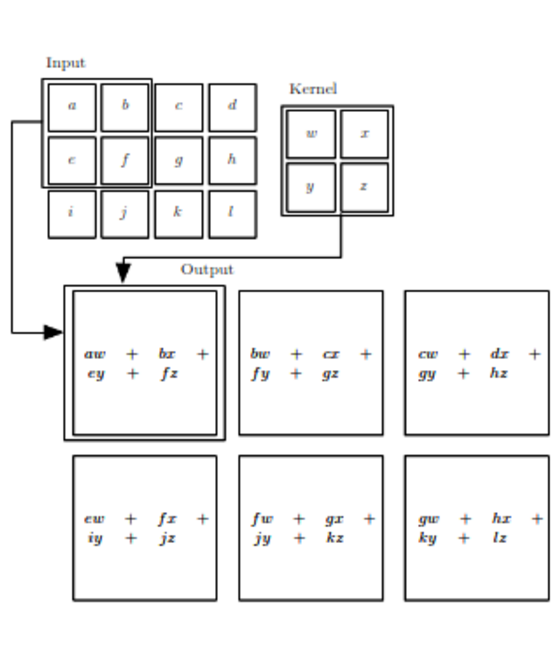
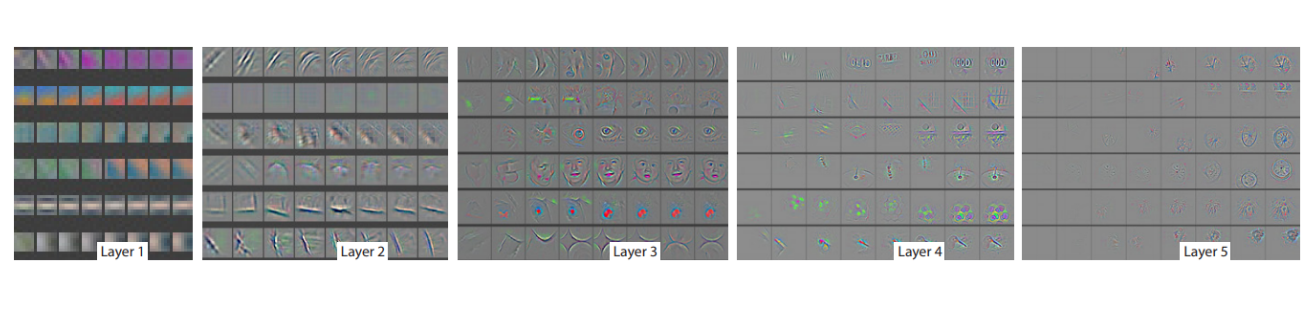


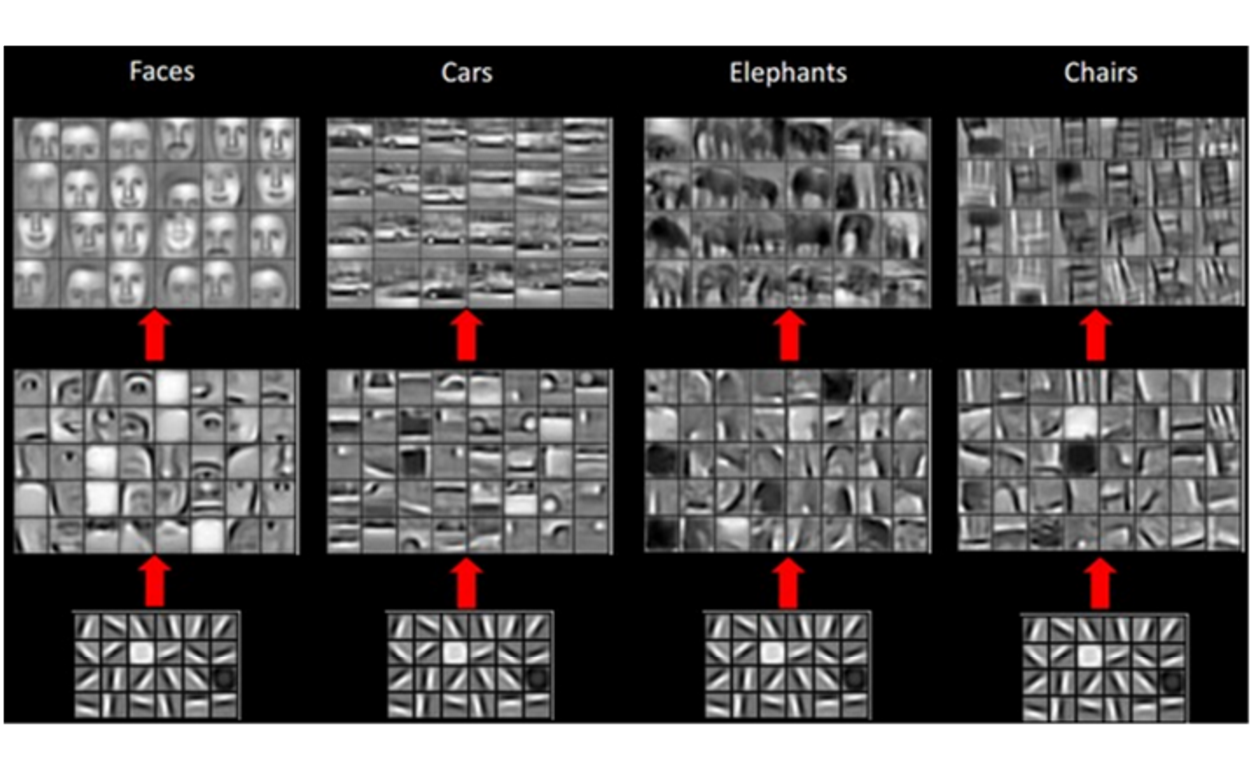
Figure 4 Convolution [12]

- Activation layer is the layer that introduces non-linearity. ReLU (Recitfied Linear Unit), introduced by Hahnloser et al[11] which simply output the same value when it is greater than 0 and output 0 when it is smaller than 0, is the most popular activation function used in computer vision as it is simple yet effectively solve the vanishing gradient problem in deep neural networks.
- Pooling layer mainly serve as a mean of downsampling to reduce the size of the network. In this study, I use Max Pooling which outputs the max value of the filter. Global Average Pooling is used before the final FC layer. It turns the entire feature map (such as the output in Figure 3) into a single value by averaging the numbers.
- Dropout, raised by Hinton et al[10], is a process that randomly switch off the output of some feature maps forcing the network to learn redundant features which is proven a very good approach to make the model more robust to variations.

The first CNN layer extracts simple patters such as edges. Following layers extract more abstract patterns from the previous layers [4].



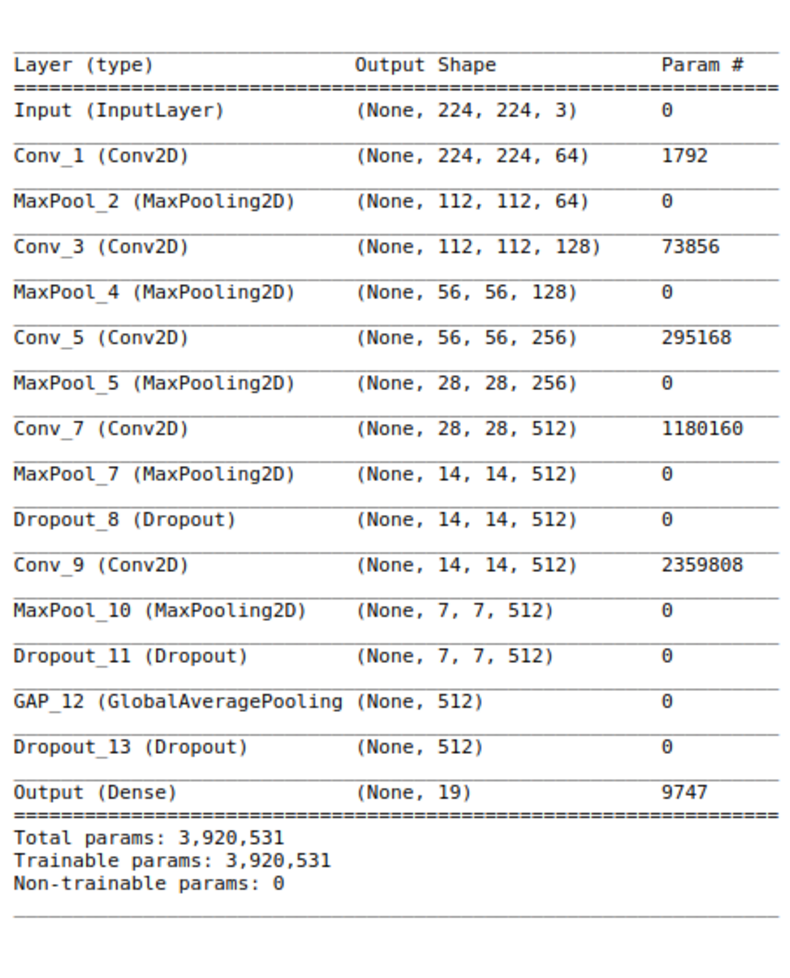
The first few layers extract relatively simple patterns which are roughly the same for different image classification tasks. For instance, basic patterns like edges and curves are the basic elements of every image [2] (Lee et al 2009).



By reusing the less abstract and less task oriented layers pre-trained by massive data sets, we transfer the knowledge learned from a different tasks to a new task. Transfer learning allows us to reuse the less abstract layers of a pre-trained model to achieve a different goal by training the more abstract/top layers with way less data and effort due to the massive reduction of trainable parameters.

Benchmark

The benchmark model - a CNN model with 5 Convolutional layers with similar architecture to VGG19 except that every block has only one CNN layer and Global Average Pooling is used instead of Fully Connected layer to reduce overfitting.



```
Layer (type)                 Output Shape              Param #
=================================================================
Input (InputLayer)           (None, 224, 224, 3)       0
Conv_1 (Conv2D)              (None, 224, 224, 64)      1792
MaxPool_2 (MaxPooling2D)     (None, 112, 112, 64)      0
Conv_3 (Conv2D)              (None, 112, 112, 128)     73856
MaxPool_4 (MaxPooling2D)     (None, 56, 56, 128)       0
Conv_5 (Conv2D)              (None, 56, 56, 256)       295168
MaxPool_5 (MaxPooling2D)     (None, 28, 28, 256)       0
Conv_7 (Conv2D)              (None, 28, 28, 512)       1180160
MaxPool_7 (MaxPooling2D)     (None, 14, 14, 512)       0
Dropout_8 (Dropout)          (None, 14, 14, 512)       0
Conv_9 (Conv2D)              (None, 14, 14, 512)       2359808
MaxPool_10 (MaxPooling2D)    (None, 7, 7, 512)         0
Dropout_11 (Dropout)         (None, 7, 7, 512)         0
GAP_12 (GlobalAveragePooling (None, 512)               0
Dropout_13 (Dropout)         (None, 512)               0
Output (Dense)               (None, 19)                9747
=================================================================
Total params: 3,920,531
Trainable params: 3,920,531
Non-trainable params: 0
```

A simpler model tend to easily overfit(or underfit if too simple) while a deeper model cannot learn due to the limitation of the insufficient number of training data.

III. Methodology

Data Preprocessing

I randomly splitting the data into training, validation and test set containing 1460, 180 and 100 images respectively. Leaving us merely 78 images on average per class which is far from sufficient to train a CNN model from stretch to classify such a diverse dataset. Add another 40 images from Google to the test set (140 images in total).

Images are loaded as 224x224 pixels images. Figure 5 shows the images loaded.

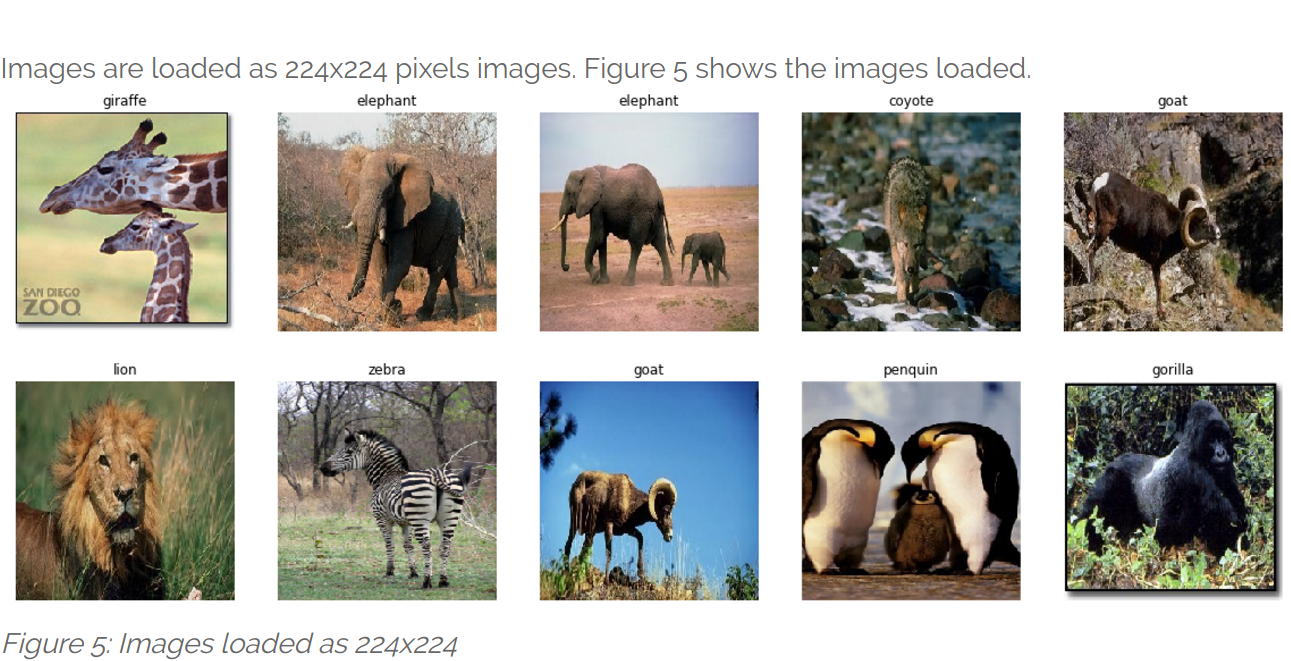


Figure 5: Images loaded as 224x224

All image pixels are scaled by 1/255 when being loaded into 4D tensor.

Image Augmentation is applied to training data to combat overfitting and to make the models more robust. I apply max 10 degrees image rotation, max 0.2 shear and zoom and chances of horizontal flip.

```
datagen = ImageDataGenerator(
    rotation_range=10,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
)
```

Code Snippet 1: Image Augmentation

Figure 6 shows the augmented images.

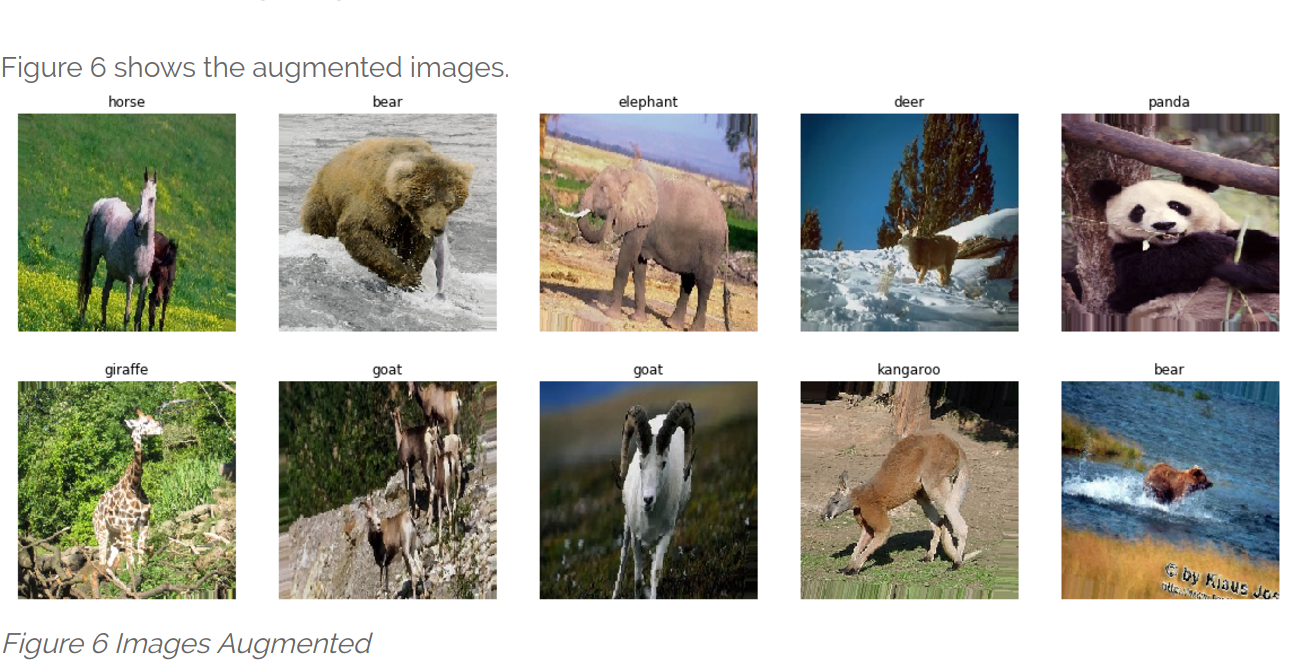


Figure 6 Images Augmented

Implementation

The transfer learning model is a VGG19 with weights pre-trained on ImageNet dataset with top layers(two Fully Connected layers) replaced with a top model.

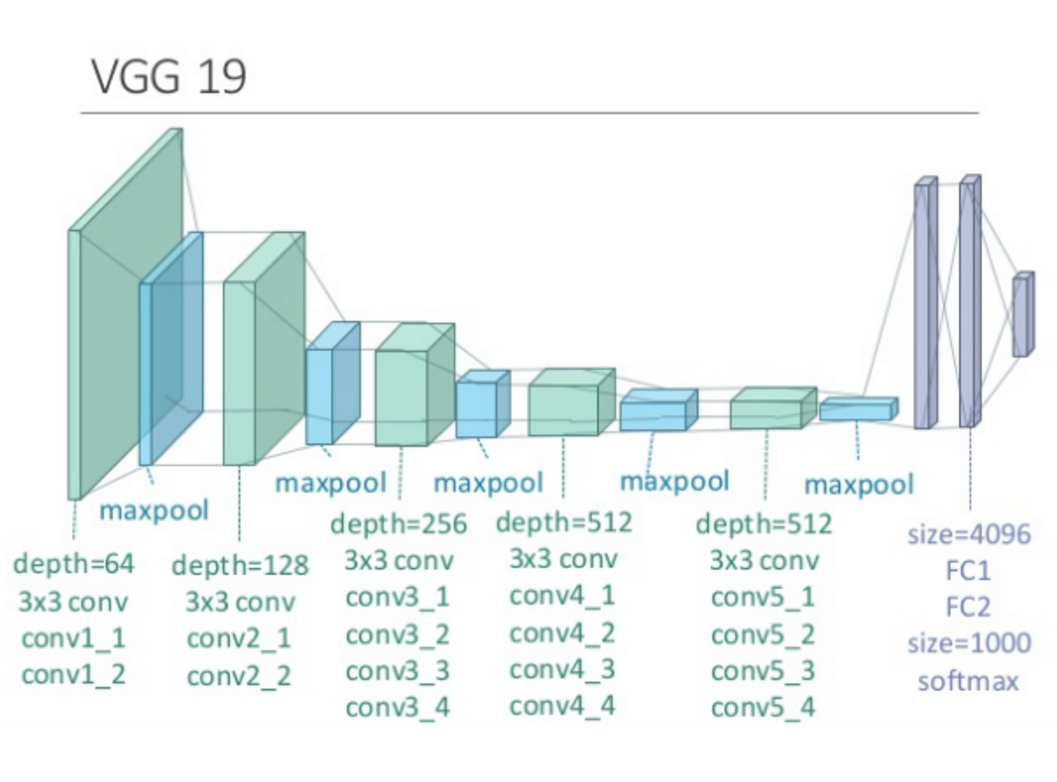
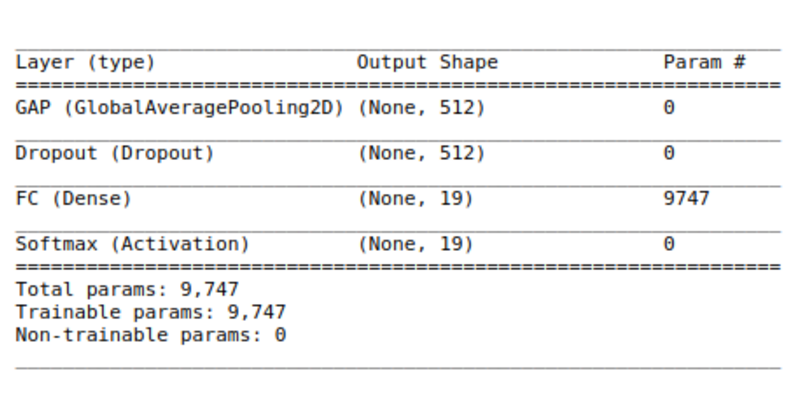


Figure 7 VGG19

Top Model in Figure 8 is used to replace the two FC layers and the output layer (in purple in Figure 7). 0.7 dropout rate is used in Dropout layer.



```
Layer (type)                 Output Shape              Param #
=================================================================
GAP (GlobalAveragePooling2D) (None, 512)               0
Dropout (Dropout)            (None, 512)               0
FC (Dense)                   (None, 19)                9747
Softmax (Activation)         (None, 19)                0
=================================================================
Total params: 9,747
Trainable params: 9,747
Non-trainable params: 0
```

Figure 8 Top Model

To train the top model in Figure 5, we need to pass the training, validation and test dataset through the VGG19(without top layers included) to obtain the input features (bottleneck features) for the top model.

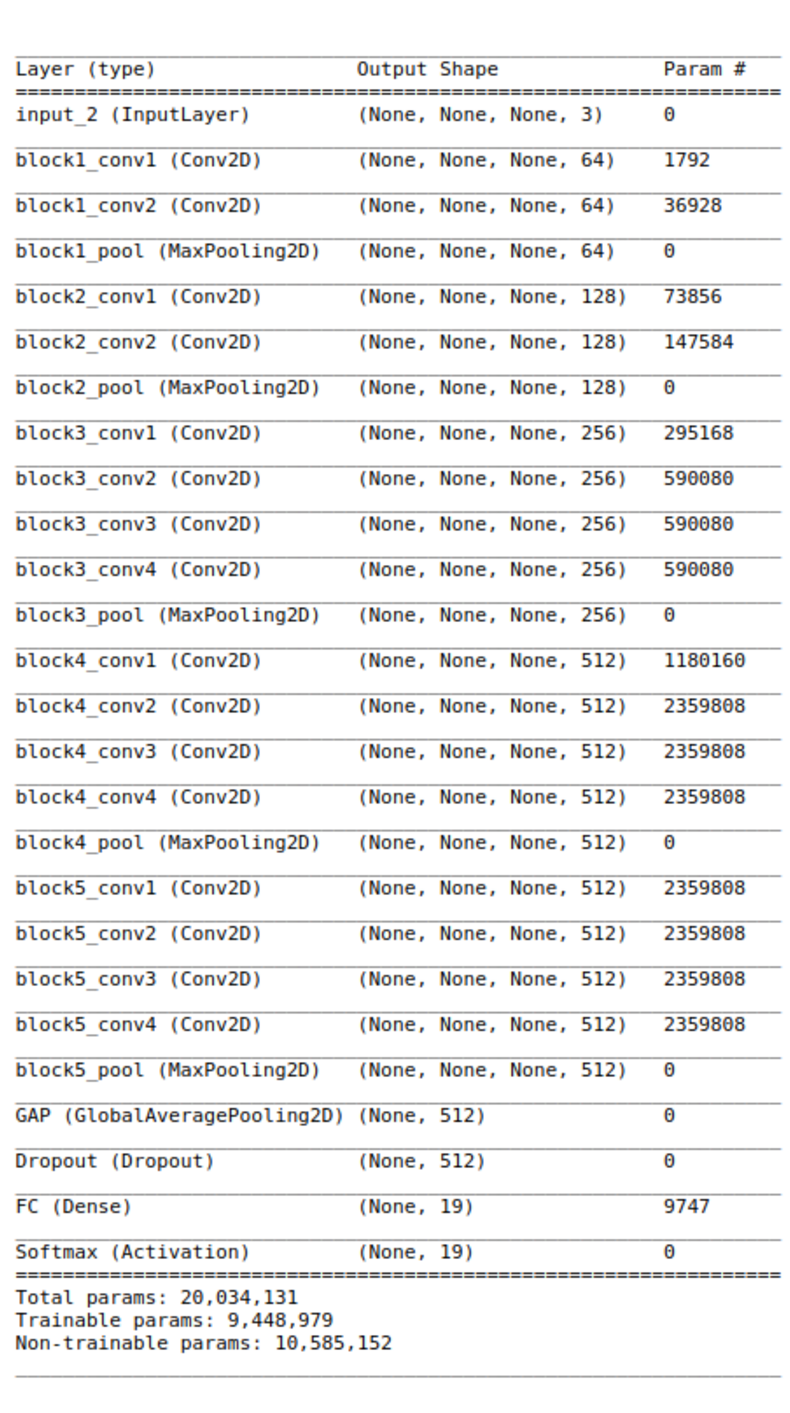
```
# Load pre-trained VGG19
pretrained_model = applications.VGG19(include_top=False, weights='imagenet')

# Generator for augmented training data
train_X_generator = datagen.flow(X_train, batch_size=batch_size, shuffle=False)

# Generate bottleneck features
bottleneck_features_train = pretrained_model.predict_generator(train_X_generator,
steps=len(X_train) / batch_size)
bottleneck_features_valid = pretrained_model.predict(X_valid)
bottleneck_features_test = pretrained_model.predict(X_test)
```

Code Snippet 2: Get bottleneck features

I then use the bottleneck features as the input to train the weight of the top model.

Refinement

After training the weights of the top model, I connect the top model trained earlier to the pre-trained VGG19(with top layers removed) to build a new model.



```
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, None, None, 3)     0
block1_conv1 (Conv2D)        (None, None, None, 64)    1792
block1_conv2 (Conv2D)        (None, None, None, 64)    36928
block1_pool (MaxPooling2D)   (None, None, None, 64)    0
block2_conv1 (Conv2D)        (None, None, None, 128)   73856
block2_conv2 (Conv2D)        (None, None, None, 128)   147584
block2_pool (MaxPooling2D)   (None, None, None, 128)   0
block3_conv1 (Conv2D)        (None, None, None, 256)   295168
block3_conv2 (Conv2D)        (None, None, None, 256)   590080
block3_conv3 (Conv2D)        (None, None, None, 256)   590080
block3_conv4 (Conv2D)        (None, None, None, 256)   590080
block3_pool (MaxPooling2D)   (None, None, None, 256)   0
block4_conv1 (Conv2D)        (None, None, None, 512)   1180160
block4_conv2 (Conv2D)        (None, None, None, 512)   2359808
block4_conv3 (Conv2D)        (None, None, None, 512)   2359808
block4_conv4 (Conv2D)        (None, None, None, 512)   2359808
block4_pool (MaxPooling2D)   (None, None, None, 512)   0
block5_conv1 (Conv2D)        (None, None, None, 512)   2359808
block5_conv2 (Conv2D)        (None, None, None, 512)   2359808
block5_conv3 (Conv2D)        (None, None, None, 512)   2359808
block5_conv4 (Conv2D)        (None, None, None, 512)   2359808
block5_pool (MaxPooling2D)   (None, None, None, 512)   0
GAP (GlobalAveragePooling2D) (None, 512)               0
Dropout (Dropout)            (None, 512)               0
FC (Dense)                   (None, 19)                9747
Softmax (Activation)         (None, 19)                0
=================================================================
Total params: 20,034,131
Trainable params: 9,448,979
Non-trainable params: 10,585,152
```

Figure 9 final model

I freeze the first 4 CNN blocks of VGG19 to fine tune the last (5th) block of VGG19 as well as our top model(Figure 8). By freezing the first 4 CNN blocks, I freeze the hard learned pre-trained weights applicable to general image classification task. Fine tuning the most abstract layers, the 5th CNN block, so that it learns the high level features specific to this task.

```
model = Sequential()

# Add VGG19 (without top layers)
for l in pretrained_model.layers:
    model.add(l)

# Add my Top Model
for l in top_model.layers:
    model.add(l)

## Lock layers until the last ConvNet block
lock_until = 17
for n, layer in enumerate(model.layers):
    if n < lock_until:
        layer.trainable = False
    else:
        layer.trainable = Tru
```

Code Snippet 3: Freeze layers

IV. Result

Model Evaluation and Validation

Benchmark Model

It took 259.6 seconds, 29 epochs to finish training the benchmark model. 8.95 seconds per epoch. On test dataset, it achieves 57% accuracy, 0.5479 kappa score and 1.83 cross-entropy loss at the 23th epoch.

	epoch	metric	train	valid	test
46	23	loss	1.117020	1.356680	1.826603
47	23	acc	0.659589	0.627778	0.571429

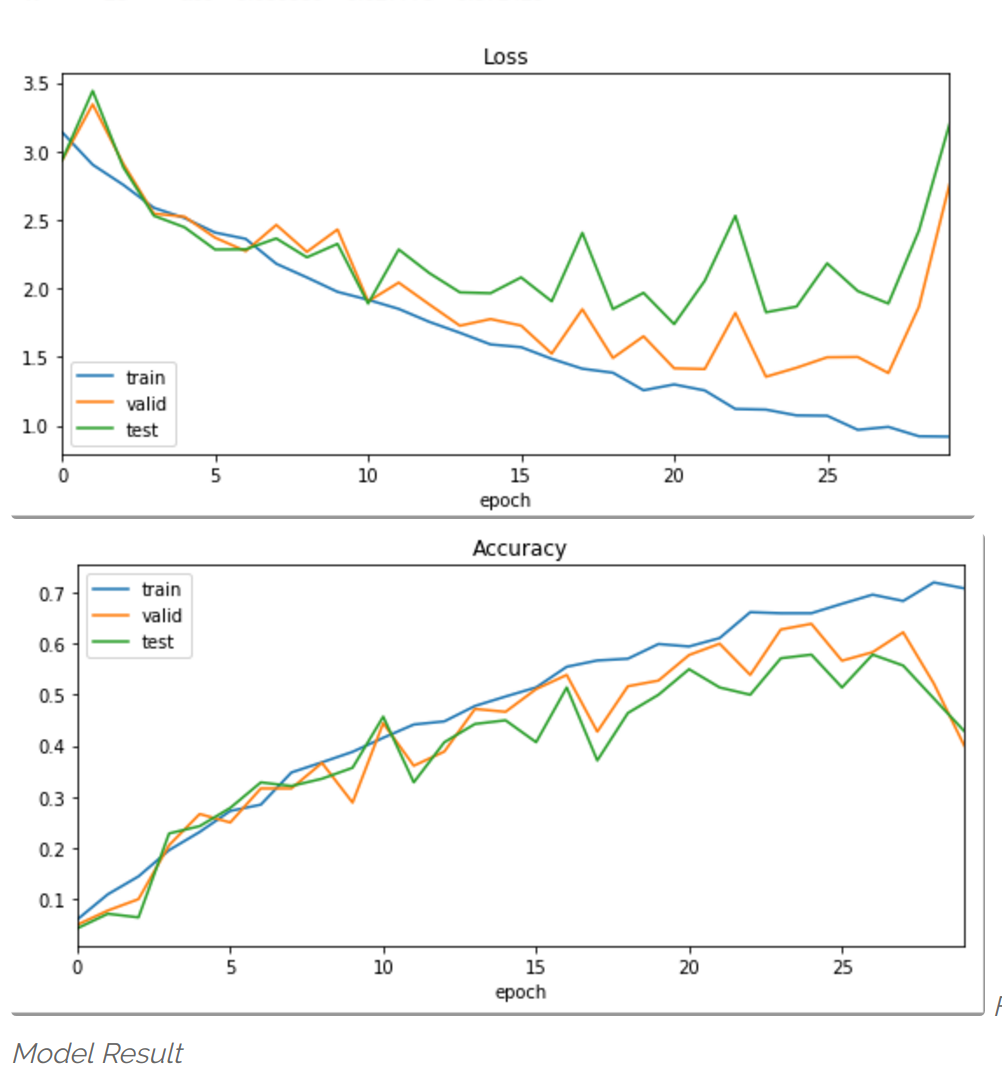


Figure 10: Benchmark Model Result

Top Model

It took 40.83 seconds, 235 epochs to finish training the top model. Merely 0.17 second per epoch. Achieve 60% test accuracy, 0.5777 kappa score and 1.41 log loss at the 224th epoch.

	epoch	metric	train	valid	test
448	224	loss	1.469626	1.365698	1.410381
449	224	acc	0.547260	0.588889	0.600000

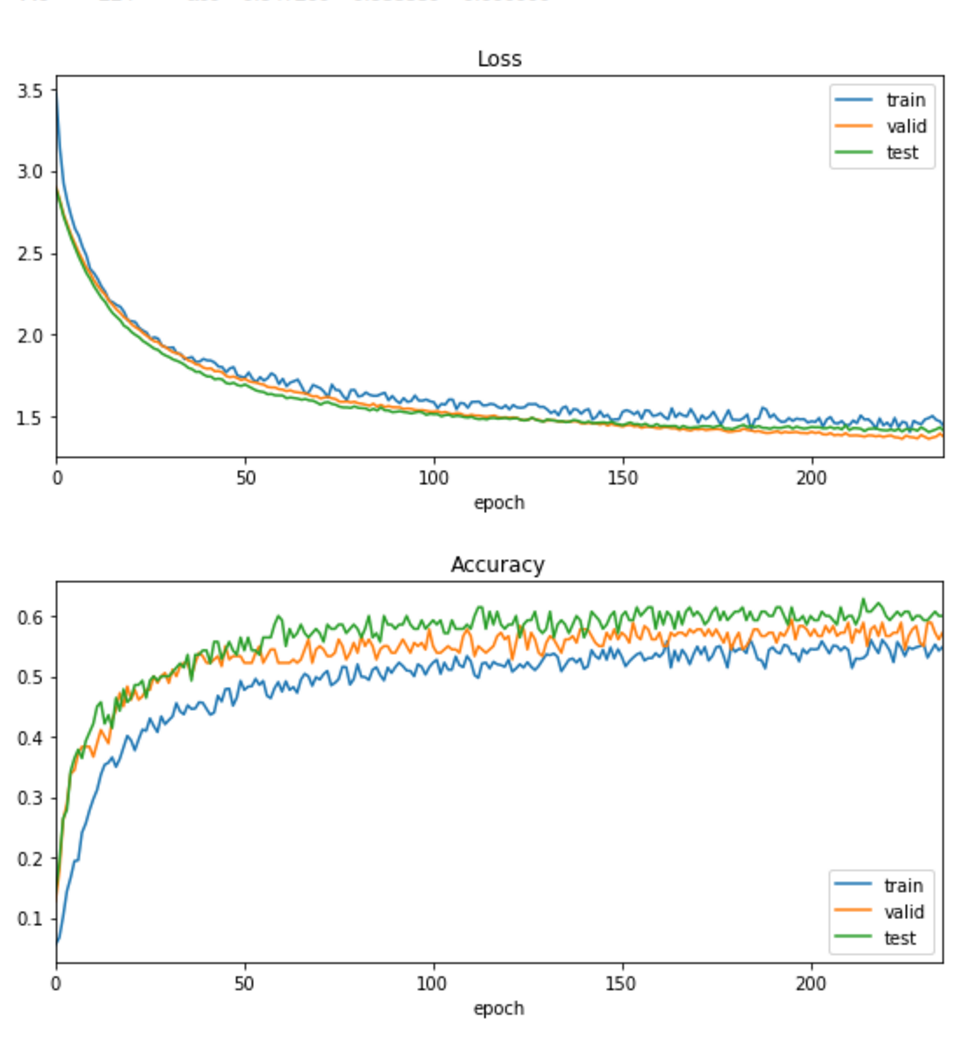


Figure 11: Top Model Result

Fine-Tuned Model

It took 375.82 seconds, 30 epochs to finish fine tuning the model. 12.5 seconds per epoch. Achieve 80% test accuracy, 0.7890 kappa score and 0.93 log loss at the 24th epoch. Another +20% improvement in test accuracy. It's impossible to train a model from stretch to achieve 80% test accuracy with so little date. Transfer learning making it all possible with the expense of more computation time (varies depends on the benchmark and transfer model complexity).

	epoch	metric	train	valid	test
48	24	loss	0.360831	0.577308	0.929925
49	24	acc	0.874658	0.811111	0.800000

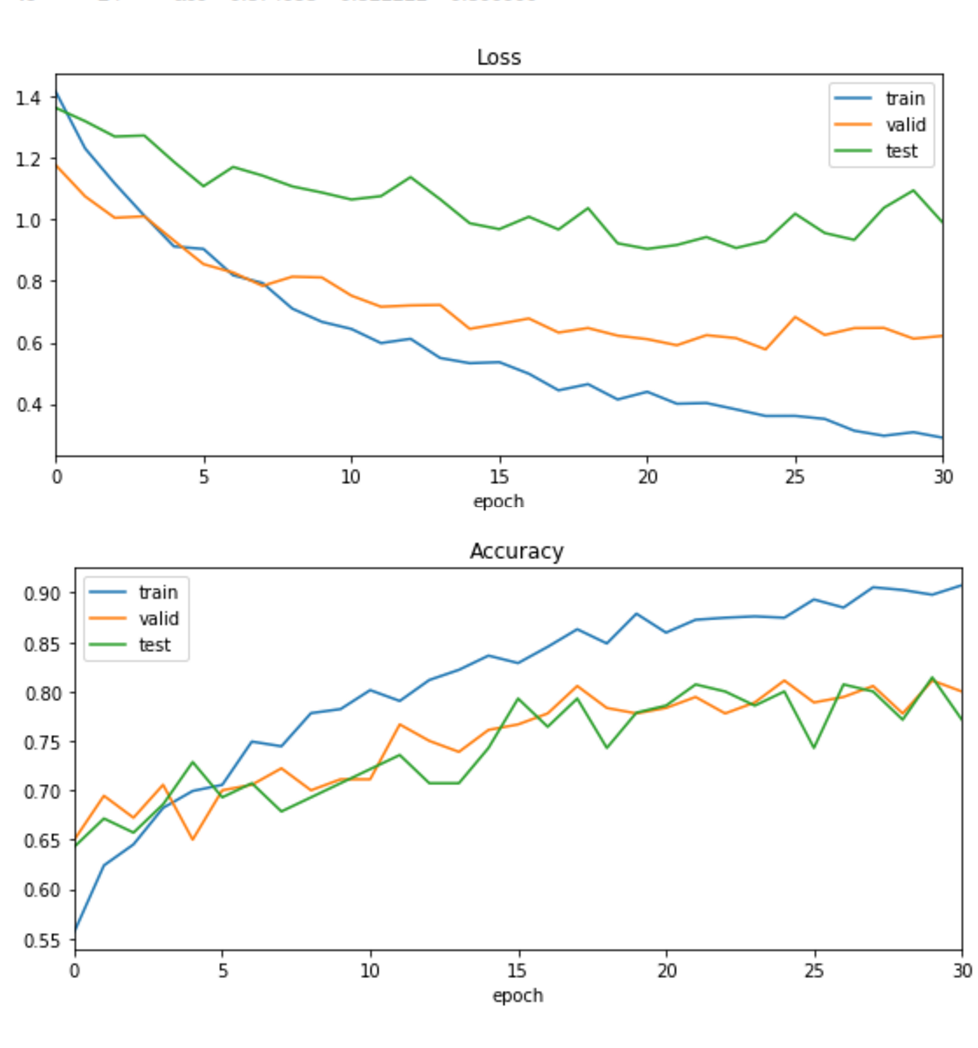


Figure 12: Fine-Tuned Model Result

Justification

As shown in Table 1, the power of transfer learning is already showing in the top model. Training the top model is 53x faster while achieving a better result. The fine-tune model significantly improve the model in all metrics.

Metrics on Test Dataset	Training Time(sec)/Epoch	Cross-Entropy Lost	Accuracy	Kappa Score
Benchmark Model	8.95	1.83	57%	0.5479
Top Model	0.17	1.41	60%	0.5777
Find-Tuned Model	12.5	0.93	80%	0.7890

Table 1 Model Evaluation

V. Conclusion

Free-Form Visualization

I use Global Average Pooling(GAP) instead of Fully Connected(FC) layers not only to combat overfitting also because it brings an interesting byproduct - weights of the last CNN can be used for nearly effortless object localization. [7][8]

The idea behind is simple. When using GAP layer before the final FC layer. The weights (512x1 vector - blue dotted lines below) connecting the GAP to each of the output node, e.g. panda, tell us which of those 512 feature maps activates the output the most. And each of the 512 nodes in GAP layer is calculated averaging a 7x7 feature map, green dotted lines below.

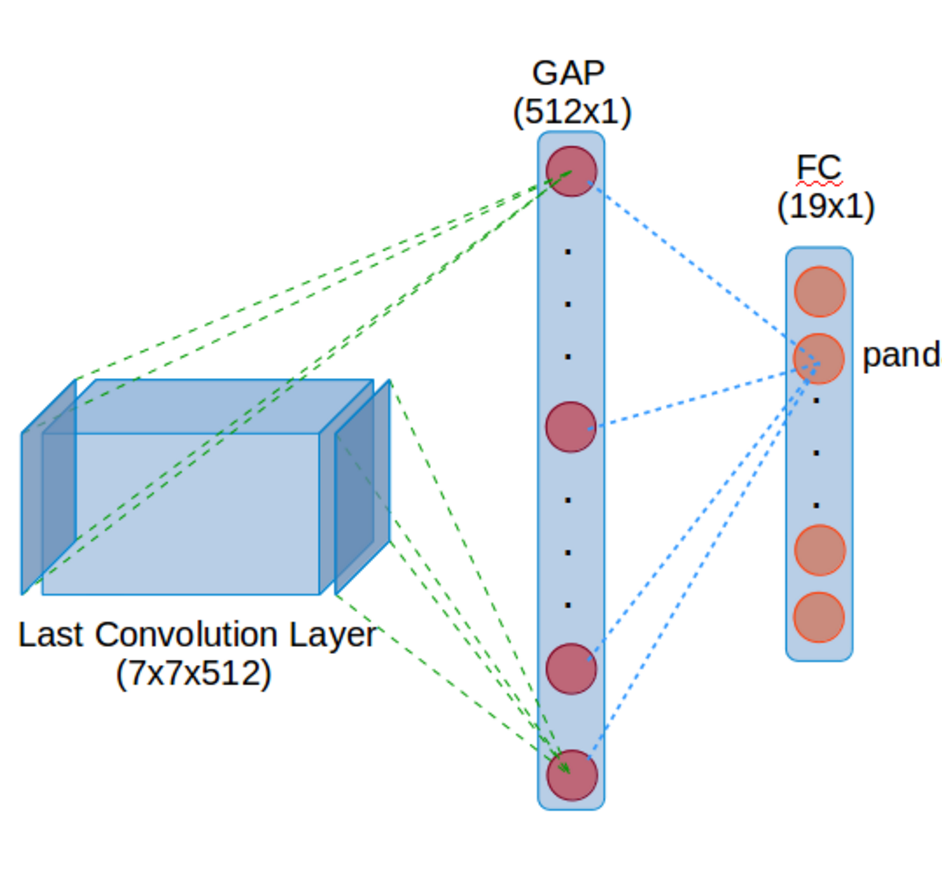


Figure 13: Zoom into the last CNN - GAP - FC layers

Therefore multiplying 7x7x512 by 512x1 gives us the weighted feature map - the activation map which allow us to visualize which part of the image is activating that particular output class.

I randomly picked 15 classes and download pictures for each class. Figure 14 shows that the model correctly classified all new images and also showing what the CNN is looking at when predicting the class label. For example, The elephant's body and nose; The patten on the body of zebra and leopard; The black and white of panda, etc. These are strong evidences the model is learning to identify animals by identifying them by their key difference from the others.

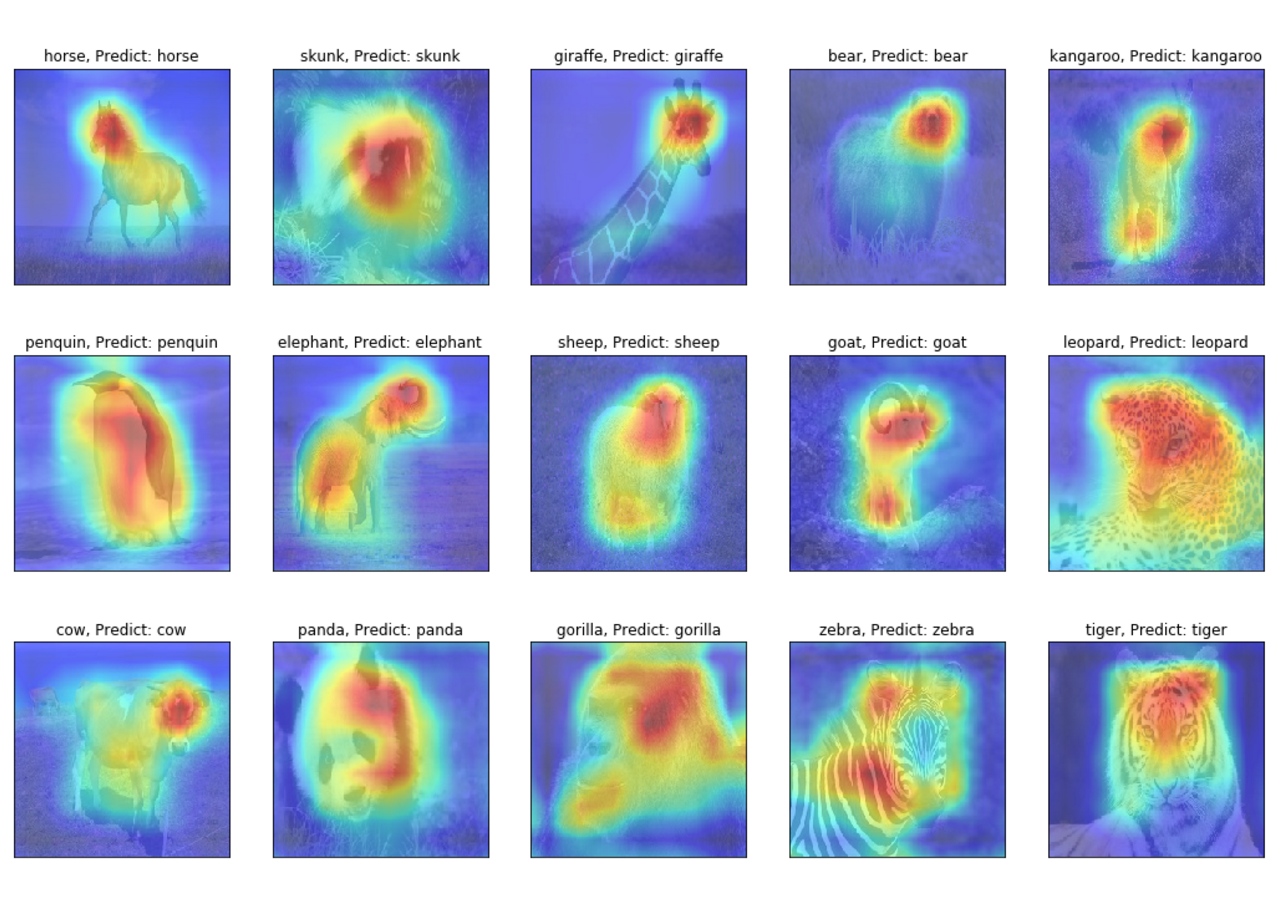


Figure 14: CNN object localization

Reflection

This study demonstrated how powerful transfer learning is by outperforming a non-trivial CNN model trained from stretch with just a fraction of the training time yet achieve a better result when the training data is insufficient. And how pre-trained model can be further fine-tuned to obtain massive performance improvement that could not otherwise be achievable training a new CNN model when longer training time is acceptable.

I can see huge potentials applying transfer learning in real life applications especially in testing out new ideas quickly without having to collect a huge amount of train data and a long training time.

I have also demonstrated an interesting byproduct of CNN model when using GAP layer between the last Convolutional layer and the FC output layer. The activation map serves as both object localization as well as a mean of model visualization which shows me how the model makes prediction.

Improvement

The final model can be further improved by unfreezing more Convolutional layers in the fine-tune process to further tweak the abstract layers towards our goal. It will produce better result than our final model however it will take substantially longer to train. And there will be a point where unfreezing more layers will not give us better result due to the insufficient of training data.

When dealing with images with multiple objects, we could use the top N highest scored (not just the best) FC weights to build the activation map to obtain the location of objects recognized by the model. Then further feed the sub-images containing the objects through the CNN again to obtain the classes for all bounding boxes. Other studies have been made to utilize this method for more robust object localization, such as the study by Singh and Lee [9] where they expand the bounding box by randomly turning off parts of the training images to force the network to learn other parts of the object (not just the most iconic part like the panda face in our example).

Reference

[1] http://yann.lecun.com/exdb/publis/pdf/lecun-99.pdf
[2] https://www.cs.princeton.edu/~rajeshr/papers/icml09-ConvolutionalDeepBeliefNetworks.pdf
[3] https://www.csc.kth.se/~heydarma/Datasets.html
[4] https://arxiv.org/pdf/1311.2901.pdf
[5] https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html
[6] https://www.kaggle.com/wiki/LogLoss
[7] http://cnnlocalization.csail.mit.edu/Zhou_Learning_Deep_Features_CVPR_2016_paper.pdf
[8] https://alexisbcook.github.io/2017/global-average-pooling-layers-for-object-localization/
[9] http://krsingh.cs.ucdavis.edu/krishna_files/papers/hide_and_seek/my_files/iccv2017.pdf [10] https://arxiv.org/pdf/1207.0580.pdf [11] R Hahnloser, R. Sarpeshkar, M A Mahowald, R. J. Douglas, H.S. Seung (2000). Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. Nature. 405. pp. 947–951. [12] http://www.deeplearningbook.org/contents/convnets.html