Object Recognition

Group 39 - Final Project

Group Members

Name	ID	Contribution
Long Thai	n10006257	25%
Phuoc Nguyen	n9951733	25%
Nguyen Gia Bao Ho	n9848967	25%
Daniel Tan	n9455001	25%

CAB420 Machine Learning | 2019 Semester 1

Table of Contents

Introduction

The goal of object recognition algorithms is to allow the computer to classify an image or identify an object(s) in an image. This is an important topic because of its vast applications in many areas. It can be used: in self driving cars to detect people, roads, signs, other cars and more in order to obey the traffic laws; for facial recognition, to quickly identify someone for searching, or use to unlock your smart device; in medicine for medical imaging, which is used for diagnosis; and so on. (Vahab, 2019)

The particular motivation of this project is to try and train a model for accurate classifications using a limited dataset. The reason for such a limitation is to experiment with methods that will have a higher training efficiency for each data point used. This will allow for more efficient models that will, in turn, be able to achieve more accurate results with smaller dataset compared to inefficient models that require more data.

Related Works

There are two common types of algorithms for object recognition. There is machine learning and neural networks. The more common approaches for machine learning is to extract features from the images in your training set and use one of the clustering methods (e.g. SVM, kNN) to do the classification. (Mathworks, n.d.) The emphasis is on the feature extraction, choosing which features will work best for classifying your dataset. The common approach for neural networks are Convolutional Neural Networks (CNN). There are two main ways of making a CNN. From scratch, where the network architecture itself is constructed and trained. Or transfer learning, where an existing trained model is used as a base and the model is trained to fine-tune it to a specific dataset. (Mathworks, n.d.)

Making a CNN from scratch takes a wealth of time and computational power to train the model. There are a couple of common approaches and variations to creating a CNN that vary the technique of analysing the image data. They are Region-CNN (RCNN); Fast and Faster RCNN; and You Only Look Once (YOLO). These CNNs try to find regions of interest to then be able to train and classify their model for efficiently and effectively. (Gandhi, 2019)

On the other hand, using a pre-trained model saves the initial time and computational expense and requires only a little more work to fine-tune the model for a specific dataset. Common pre-trained models include: Inception V3, ResNet and AlexNet. (Shao, 2019) When using a pre-trained model, the classification layer would be removed and replaced with new layers from training against the training dataset. (Marcelino, 2018)

Data

The dataset used to train our algorithms was the Caltech 256 dataset. It contains 30,607 images from 256 categories. (Caltech, 2006) From the 256 categories we chose four and they were: bear, dog, gorilla, leopard. We chose these four categories to challenge the algorithms as a bear is similar in shape and colour to a gorilla, and the leopard is similar in shape to a dog.

The dataset contains a broad range of categories, however, each category didn’t have the quantity of images required to train for all categories. By limiting the dataset, we can train a better classifier and reduce the training time for our algorithms.

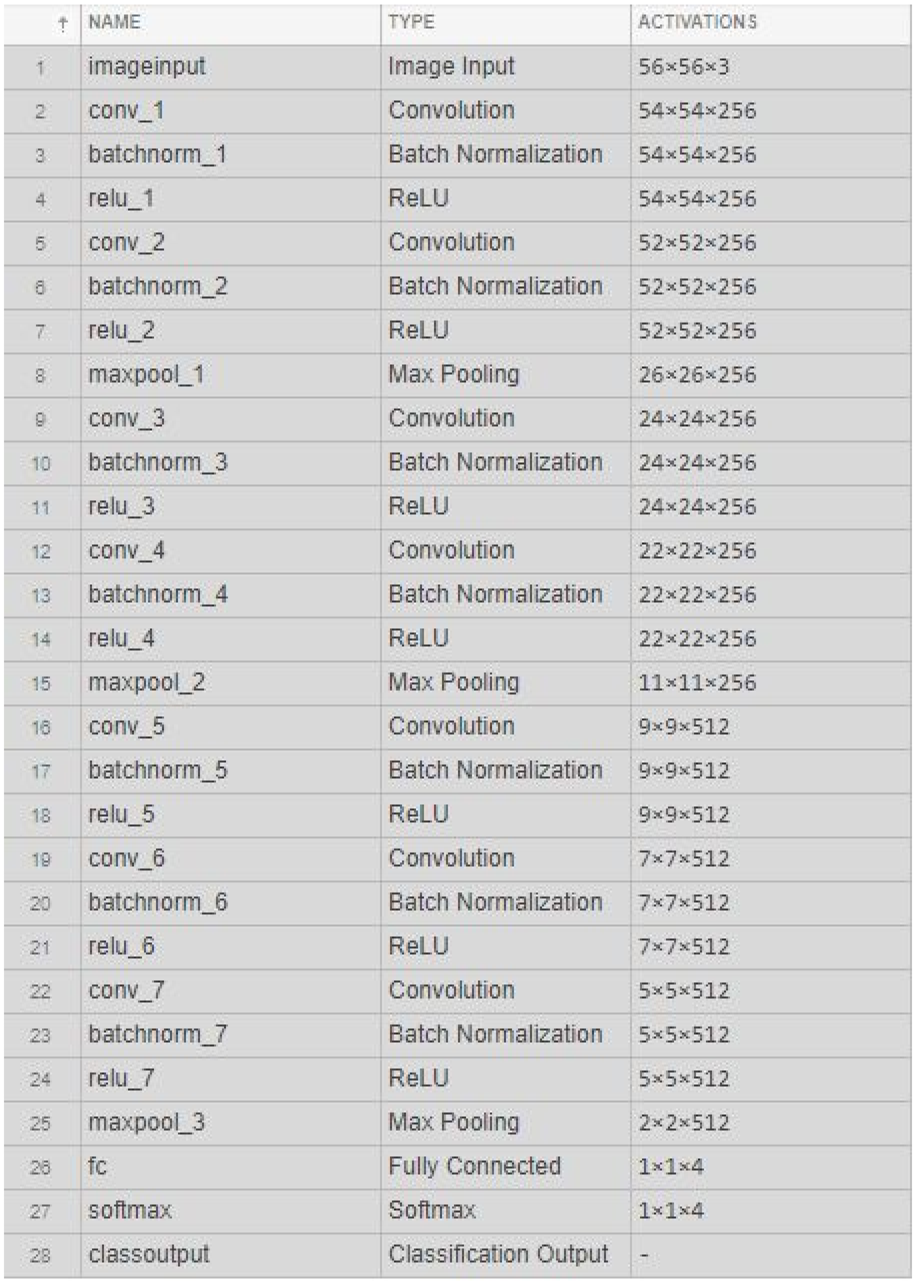
The images were resized to 64x64 pixels and 56x56 pixels in Matlab to make the images consistent for training and reduce the training time by using smaller images. These sizes were chosen as it made the images small for training, but large enough for the image to still be recognisable and of significance.

Methodology

Since we didn’t have enough computing power nor a large enough dataset, we could not train a full (CNN). Instead, we chose to do multiple approaches to a basic CNN and transfer learning.

Basic CNN 1

The first approach to classifying the dataset was using a convolutional neural network (CNN). This approach was chosen over multi-layer perceptron (MLP) due to its speed advantage. CNN reduces the amount of connections between each layer in the network which allows the network to go deeper. The CNN used in this approach was developed in MATLab and comprised of 28 layers. This allowed the network to classify images with moderate accuracy.



↑	NAME	TYPE	ACTIVATIONS
1	imageinput	Image Input	56×56×3
2	conv_1	Convolution	54×54×256
3	batchnorm_1	Batch Normalization	54×54×256
4	relu_1	ReLU	54×54×256
5	conv_2	Convolution	52×52×256
6	batchnorm_2	Batch Normalization	52×52×256
7	relu_2	ReLU	52×52×256
8	maxpool_1	Max Pooling	26×26×256
9	conv_3	Convolution	24×24×256
10	batchnorm_3	Batch Normalization	24×24×256
11	relu_3	ReLU	24×24×256
12	conv_4	Convolution	22×22×256
13	batchnorm_4	Batch Normalization	22×22×256
14	relu_4	ReLU	22×22×256
15	maxpool_2	Max Pooling	11×11×256
16	conv_5	Convolution	9×9×512
17	batchnorm_5	Batch Normalization	9×9×512
18	relu_5	ReLU	9×9×512
19	conv_6	Convolution	7×7×512
20	batchnorm_6	Batch Normalization	7×7×512
21	relu_6	ReLU	7×7×512
22	conv_7	Convolution	5×5×512
23	batchnorm_7	Batch Normalization	5×5×512
24	relu_7	ReLU	5×5×512
25	maxpool_3	Max Pooling	2×2×512
26	fc	Fully Connected	1×1×4
27	softmax	Softmax	1×1×4
28	classoutput	Classification Output	-

Transfer Learning

Transfer learning is a machine learning method that differs from the traditional convolutional neural network training methods as it reuses a pre-trained model as a starting point for another model. In this case, transfer learning was used by taking layers from AlexNet, a convolutional neural network that has trained on more than millions of images, and used them to train our CNN through the assistance of the Deep Learning Toolbox.

The process of applying transfer learning using AlexNet follows the procedure of firstly loading the pre-trained network, replacing the final layers to learn features specific to the dataset, training the network with the training images and options, predicting and assessing the network accuracy, followed by finally deploying the results.

The model was trained using stochastic gradient descent with momentum with an initial learning rate of 0.0001 and 20 epochs, where every epoch randomly shuffles the training data.

Basic CNN 2

Convolutional Neural Network is one powerful way to develop an AI that can learn to recognise objects by images. The neural network was built using Keras and Tensorflow, the libraries helps with building machine learning algorithms.

The algorithm is constructed with two convolution layers, two max pooling layers, one flatten layer, two dense layers and one dropout layer. For the convolution layers, an image of size 64 by 64 pixels is fed to the neural network. The first layer and the second layer uses 64 and 32 feature filter respectively to learn from the input image. A dense layer with 128 nodes acts as a fully connected layer. The dropout layer is added to prevent overfitting as it will randomly turn of some nodes while learning. All layers used rectifier activation function except the output layer uses softmax activation function. The algorithm uses gradient descent to optimise its learning, reduce losses and improve accuracy.

The dataset for this algorithm is split into training and testing set with the ratio of 80:20. 80% of all images is used for training and 20% is for testing.

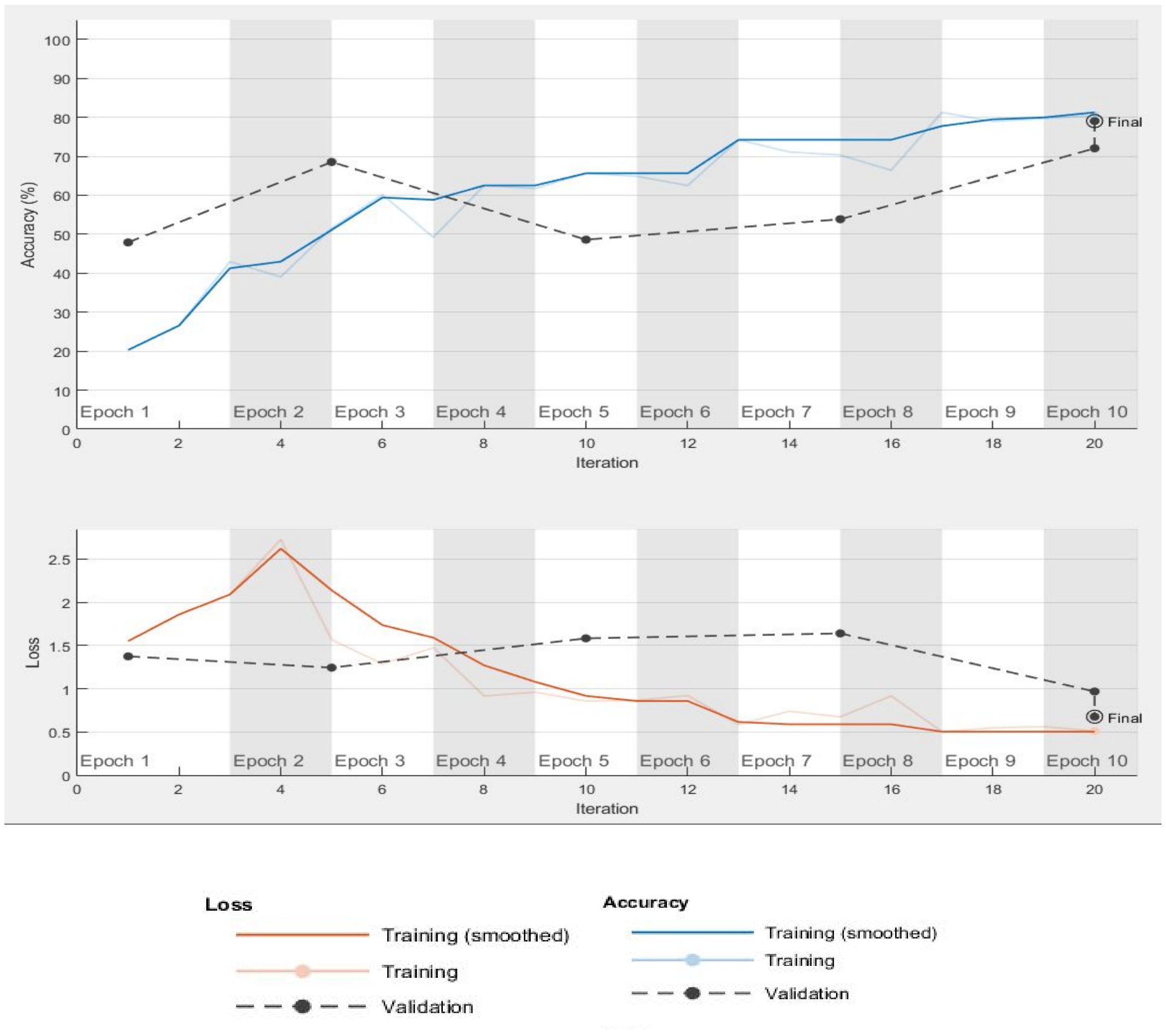
In order to successfully train a CNN, data is one of the most important factors. However, there was only 482 images for training and 120 images for testing so that data augmentation technique is used for this algorithm. For every image in the training set, the image will be flip, zoom or shear to make more images to training the neural network.

Training checkpoint is also applied for this algorithm. While training, the best model’s checkpoint is created and saved. The checkpoint can be reloaded and use for other purposes or retrain to improve performance.

Results

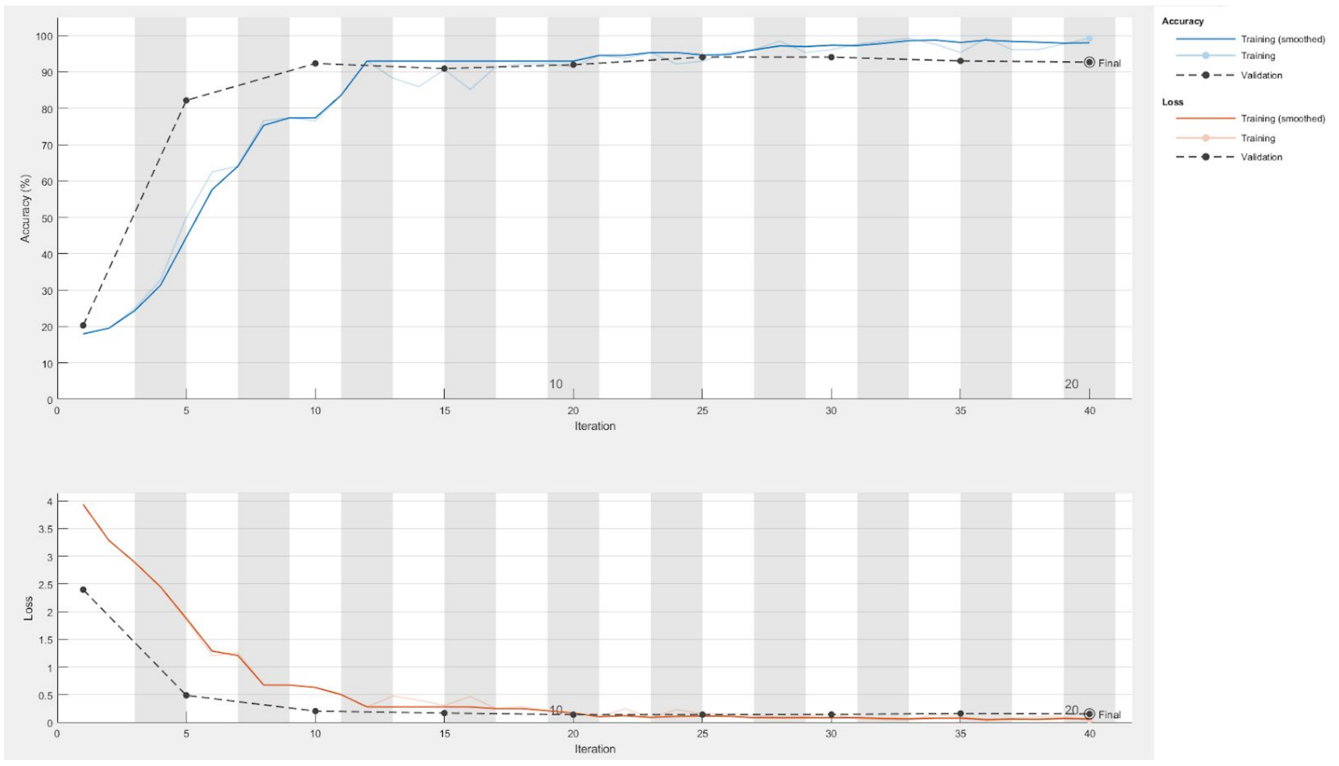
Basic CNN 1 Results

Due to the computing resources available to us, the dataset was restricted to 4 categories. The network yielded an average accuracy of approximately 79% on this dataset. The following figures show the loss and accuracy of the network throughout its training.



Transfer Learning Results

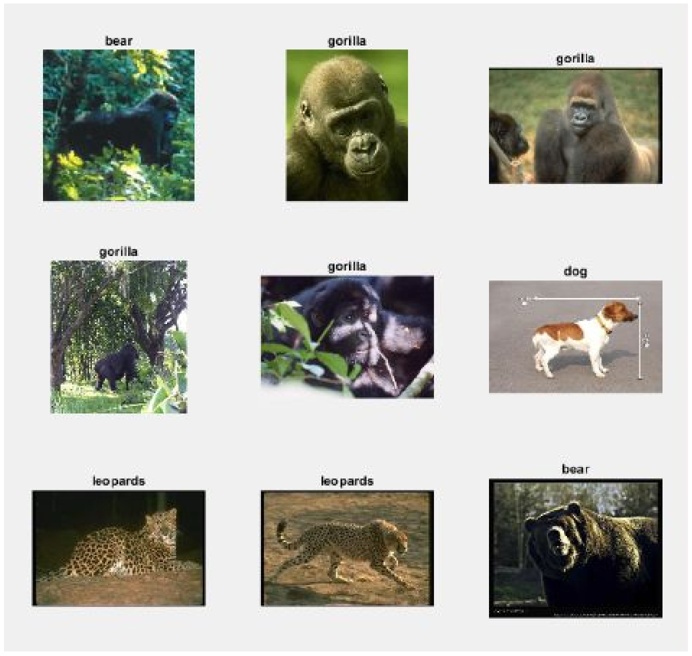
The results from transfer learning were very good. After applying transfer learning to classify the images, we achieved consistent results ranging from 92%-95% accuracy in correctly classifying the categories for each of the images. Below is a graph showing the progress of the network's training and validation accuracy when training, as well as a loss graph.



(Refer to Appendix 1: Transfer Learning Code to generate this graph)

The time spent training was approximately five minutes using the CPU (GPU was incompatible) and achieved a validation accuracy of 92.66%.

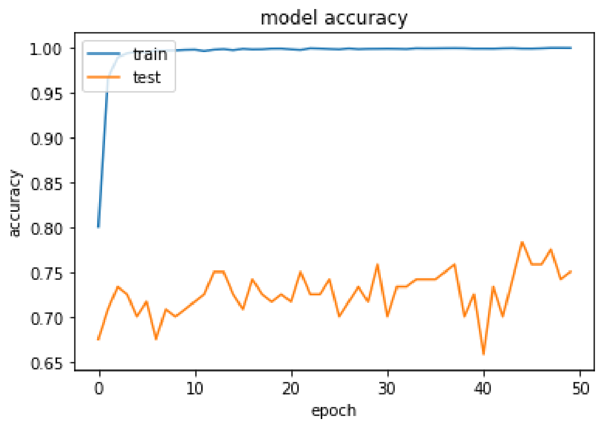
Showing the results visually, the network predicted 8 out of 9 of the categories correctly (misclassified the gorilla as a bear in the first image).

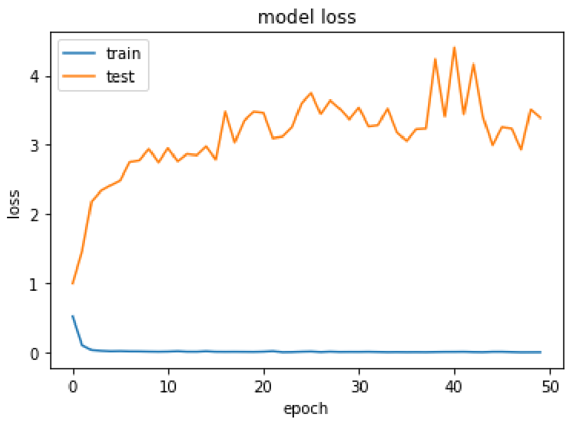


(Refer to Appendix 1: Transfer Learning Code to generate this visual representation)

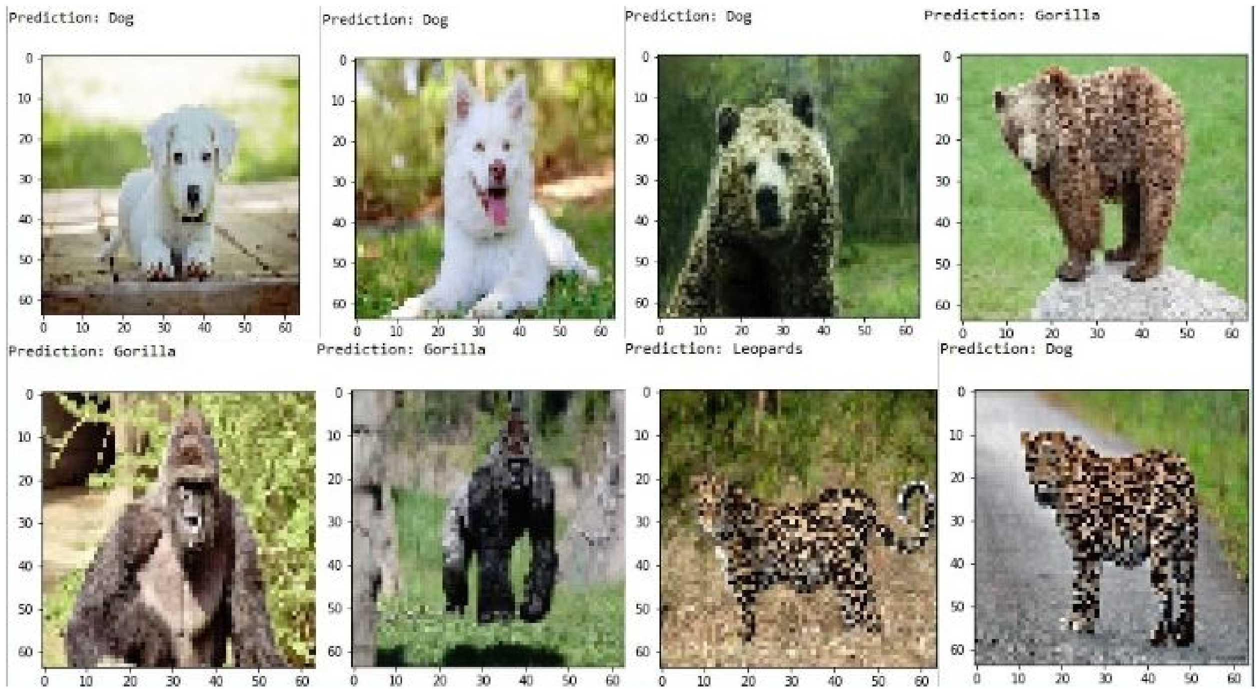
Basic CNN 2 Results

Below is the accuracy and loss of the model. The best accuracy is 78.33%. After reaching its best result, the accuracy starts to reduce. The loss does not go down during training as it should. Instead, it starts to increase. This could be that the CNN model is overfitting. There was not enough computing resources to increase the complexity of the model to make it more efficient and reduce overfitting.





The result of some example prediction is shown below. The model was very good at predicting dog and gorilla. However, it misclassified the bear as a dog or a gorilla; and the leopard as a dog.



Compare

There are many competitions in the object detection and classification space. Most recent competitions delve into object detection, using bounding boxes to classify a section of the given image; and object segmentation, where they outline the object they are classifying. Both of these types of classification are out of scope for this project, but have more utility and application as opposed to simple object classification of the entire image.

ImageNet is an image database that also hosts object detection and localization competitions. (ImageNet, n.d.) We are interested in their object localization since it is most similar to what this project aims to achieve. In 2017, a team named WMW made the top for image classification, using their new design of a building block architecture called “Squeeze-and-Excitation (SE)”. (ImageNet, 2017) Their classification error rate was 0.02251, which would be an accuracy of 97.75% accuracy. (ImageNet, 2017)

Our best accuracy results are 79% for basic CNN 1, 92.66% for transfer learning and 78.33%. Comparing our results to WMW’s results, WMW achieved a higher accuracy than any of our approaches. WMW’s algorithm was trained using ImageNet’s dataset which consisted of 1.2

million images of 1000 categories with 150,000 images being used for test and validation. (ImageNet, 2017) On the other hand, we had 606 images total across 4 categories. For ImageNet's dataset, each category would have on average 1,200 images, while our 4 categories would have 151 images on average. They also used image augmentation which would further increase the dataset. One reason for their accuracy is their large dataset and image augmentation. With more data, their CNN would be able to achieve more accurate results.

WMW also used pre-trained models as a basis for their CNN, specifically a modified version of ResNeXt for the competition. (Jie Hu, 2017) They applied their SE component to the model in order to improve upon it. Their SE component of their CNN "...adaptively recalibrates channel-wise feature responses by explicitly modelling interdependencies between channels.". (Jie Hu, 2017) This in conjunction with the pre-trained model improve the performance of the CNN compared to our simple approaches.

Conclusion

Our approaches have taken in a limited data set of four categories. This limitation isn't ideal for CNNs as they require large data sets to be used for training and this is reflected in our basic CNN approaches. However, we have achieved a reasonable result using transfer learning with 92.66% accuracy. The reason for this is that the transfer learning approach is useful for smaller data sets as the convolutional layers have already been trained using thousands of images, allowing for smaller data sets to leverage this.

Basic CNN 1 sets the benchmark for our approaches providing a moderately accurate classifier. Basic CNN 1 is based off of existing CNN architecture models (i.e. VGG16). Basic CNN 2 bases its CNN architecture from existing architectures, but also applies image augmentation to increase the number of images in the data set. This is approach provides more data to train the CNN and is especially useful for limited datasets. The results show a marginal increase in accuracy compared to Basic CNN 1. However, it is also prone to overfitting the training data as the augmented images may not differ much from each other.

Future Considerations

Keeping with the goal of creating a model that accurately classifies image using a limited dataset, we could improve our approaches by increasing the image sizes used for training our models; use CNN to extract features then apply a classification model; and apply image augmentation along with our transfer learning model.

Increasing the size of our images, the CNN will have more details in the image to extract features from. By doing so, the CNN will have more information to propagate and learn from, which could possibly increase our accuracy. This will however, drastically increase the training time depending on how large the images are reshaped to.

Combining a CNN and a machine learning classifier, for example SVM, could lead to better classification results. The CNN would be able to learn and extract unique features from training set, while the classifier will do the classifications. Many combinations of CNN models and machine learning classifiers will need to be trialed in order to find the most suitable one for our limitations.

The pre-trained model could be used in conjunction with image augmentation. Image augmentation will supply more training data to train the model and in turn further improve the accuracy for the given dataset. Depending on the augmentations, this could also lead to overfitting as seen in Basic CNN 2's results.

References

Caltech. (2006, November 15). *Caltech 256*. Retrieved from Caltech: http://www.vision.caltech.edu/Image_Datasets/Caltech256/

Gandhi, R. (2019, July 10). *R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms*. Retrieved from Towards Data Science: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

ImageNet. (2017). *Large Scale Visual Recognition Challenge 2017 (ILSVRC2017)* . Retrieved from ImageNet: http://image-net.org/challenges/LSVRC/2017/index#loc

ImageNet. (2017). *Object detection (DET)*. Retrieved from ImageNet: http://image-net.org/challenges/LSVRC/2017/results#team

ImageNet. (n.d.). *ImageNet*. Retrieved from ImageNet: http://www.image-net.org/

Jie Hu, L. S. (2017, September 5). *Squeeze-and-Excitation Networks.* Retrieved from Cornell: https://arxiv.org/abs/1709.01507

Marcelino, P. (2018, October 24). *Transfer learning from pre-trained models*. Retrieved from Towards Data Science: https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751

Mathworks. (n.d.). *Object Recognition*. Retrieved from Mathworks: https://au.mathworks.com/solutions/image-video-processing/object-recognition.html

Shao, C. (2019, April 16). *Approach pre-trained deep learning models with caution*. Retrieved from Medium: https://medium.com/comet-ml/approach-pre-trained-deep-learning-models-with-caution-9f0ff739010c

Vahab, A. (2019, April 27). *What are some interesting applications of object detection?* Retrieved from Quora: https://www.quora.com/What-are-some-interesting-applications-of-object-detection

Appendices

Appendix 1: Transfer Learning Code

```
%%% Make sure this file is in the same file path as 4_AnimalCategories
%% Get alexnet as our transfer learning network
net = alexnet;

% Initialise layers
alexnetLayers = net.Layers;

% Store images into a datastore
imagePath = '4_AnimalCategories';
imageDs = imageDatastore(imagePath,'IncludeSubfolders',true,...
    'LabelSource','foldernames');

% Split into training and validation randomly
numTraining = 80;
[imageTrain,imageValidateL] = splitEachLabel(imageDs,numTraining,'randomize');

% Change all images to size 227x227 and convert to RGB so that it can be
% used with the AlexNet layers
imageTrain = augmentedImageDatastore([227, 227, 3], imageTrain,...
    'ColorPreprocessing', 'gray2rgb');
imageValidate = augmentedImageDatastore([227, 227, 3], imageValidateL,...
    'ColorPreprocessing', 'gray2rgb');

% AlexNet's layers are for 1000 classes, configure the layers to fit our
% dataset by removing last 3 layers
newLayer = alexnetLayers(1:end-3);

% Add more layers into the end of the array
numCategories = 4;
newLayer(end+1) = fullyConnectedLayer(numCategories,'WeightLearnRateFactor',10,...
    'WeightL2Factor', 1, 'BiasLearnRateFactor', 20, 'BiasL2Factor', 0);
newLayer(end+1) = softmaxLayer;
newLayer(end+1) = classificationLayer();

% Set training options
trainingOpts = trainingOptions('sgdm', ...
    'InitialLearnRate',0.0001, ...
    'MaxEpochs',20, ...
    'Shuffle','every-epoch', ...
    'ValidationData',imageValidate, ...
    'ValidationFrequency', 5, ...
    'Verbose', false, ...
    'Plots','training-progress');

% Train the network
CNN = trainNetwork(imageTrain, newLayer, trainingOpts);

%% Testing the classifier visually
[predictY, validateScore] = classify(CNN, imageValidate);

idx = randperm(numel(imageValidate.Files), 9);
figure
for j = 1:9
    subplot(3,3,j)
    image = readimage(imageValidateL, idx(j));
    imshow(image)
    classifiedLabel = predictY(idx(j));
    title(string(classifiedLabel));
end
%% Test accuracy of classifier
validateY = imageValidateL.Labels;
accuracy = sum(predictY == validateY)/numel(validateY)
```

Appendix 3: Basic CNN 2 code

```
# -*- coding: utf-8 -*-


# Import keras libraries and packages
import keras
from keras.models import Sequential
from keras.layers import Convolution2D, MaxPooling2D, Flatten, Dense, Dropout
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ModelCheckpoint
import numpy as np
from keras.preprocessing import image

# Initializing the CNN
classifier = Sequential()

# Convolution
classifier.add(Convolution2D(filters=64, kernel_size=3, input_shape=(64, 64, 3),
activation='relu'))

# Pooling
classifier.add(MaxPooling2D(pool_size=(2, 2)))

# Convolution
classifier.add(Convolution2D(filters=32, kernel_size=3, activation='relu'))

# Pooling
classifier.add(MaxPooling2D(pool_size=(2, 2)))

# Flattening
classifier.add(Flatten())

# Full connected layer
classifier.add(Dense(units=128, activation='relu'))
classifier.add(Dropout(0.5))
classifier.add(Dense(units=4, activation='softmax'))
```

```
# Compiling
classifier.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])

# Start training
# Image preprocessing

train_datagen = ImageDataGenerator(
    rescale=1. / 255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    vertical_flip=True
    )

test_datagen = ImageDataGenerator(rescale=1. / 255)

training_set = train_datagen.flow_from_directory(
    'dataset/training_set',
    target_size=(64, 64),
    batch_size=32,
    class_mode='categorical')

test_set = test_datagen.flow_from_directory(
    'dataset/test_set',
    target_size=(64, 64),
    batch_size=32,
    class_mode='categorical')


# checkpoint
filepath="weights.best.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_acc', verbose=1, save_best_only=True,
mode='max')
callbacks_list = [checkpoint]



# fit to model
history = classifier.fit_generator(
    training_set,
    steps_per_epoch=486,
    epochs=250,
    validation_data=test_set,
    validation_steps=120,
    callbacks=callbacks_list
    )
```

```
import matplotlib.pyplot as plt

print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# reload the best weights
classifier.load_weights("weights.best.hdf5")
# Compiling
classifier.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])



# Making prediction

test_image = image.load_img('dataset/prediction_images/leopards-1.jpg', target_size=(64,
64))
plt.imshow(test_image)

test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, 0)

print("\n\n")
pred = classifier.predict(test_image)

if pred[0][0] == 1:
   print('Prediction: Bear')
elif pred[0][1] == 1:
   print('Prediction: Dog')
elif pred[0][2] == 1:
   print('Prediction: Gorilla')
```

```
elif pred[0][3] == 1:
    print('Prediction: Leopards')
```