Object Recognition

The Problem

- Given an image, a computer is not able to recognise what is in that image.
- We want the computer to be able to determine what is in an image for various applications.

Applications

Applications for object recognition include:

- Identifying stop signs for self driving cars
- Finding Wally
- Character Recognition to read text
- Face Recognition to identify faces

The Motivation

- Increase classification accuracy using less data
- Learn from other solutions to create our own object classification

Dataset

- We used the Caltech dataset where we took 4 categories to classify

Methodology

Pre-processing

- Resized images to be square and consistently sized
- Limited to 4 categories from the dataset
  - Bear
  - Dog
  - Leopard
  - Gorilla

Algorithm 1 - Overview

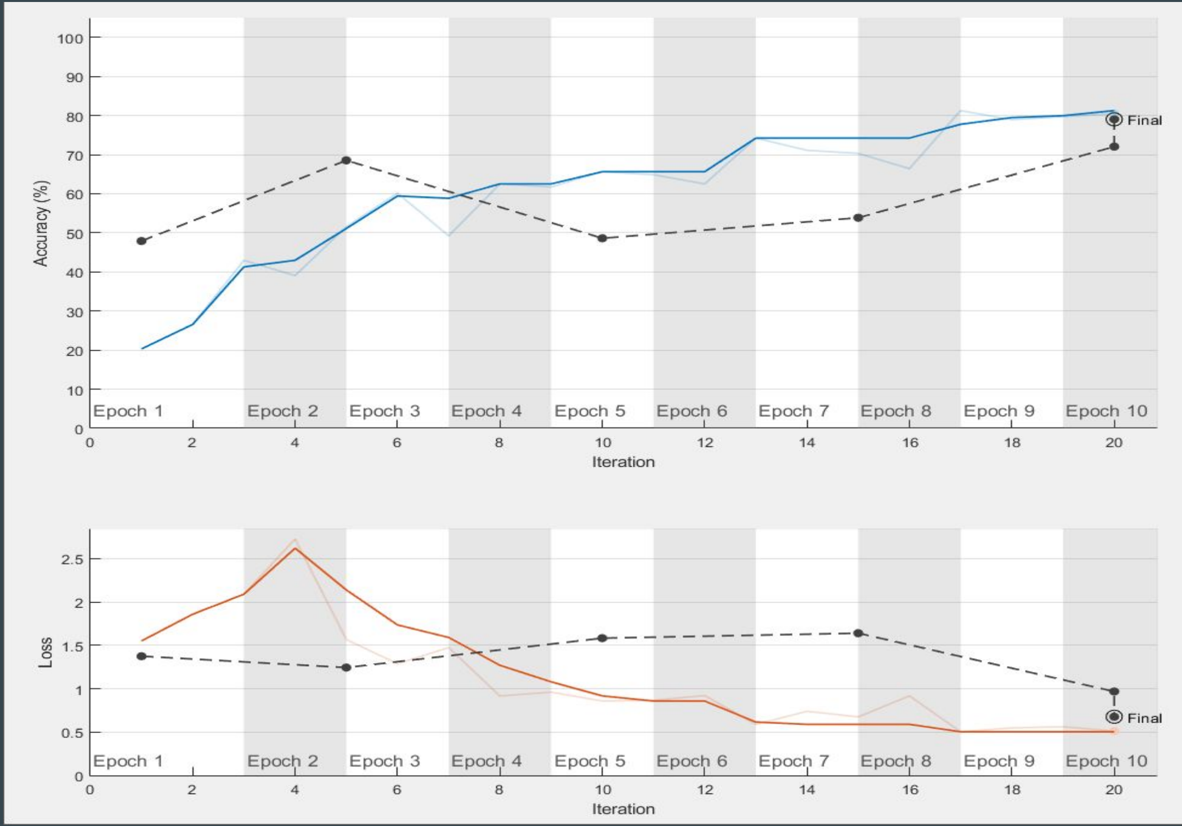
- A standard CNN
- Input size: 56x56x3
- Architecture inspired by VGG16 architecture
- Epochs: 10
- Accuracy: ~79%

Algorithm 1 - Architecture

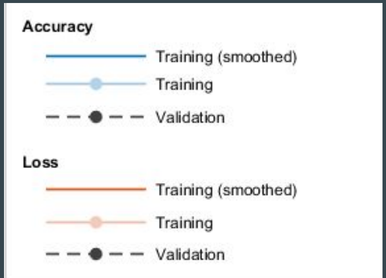
	NAME	TYPE	ACTIVATIONS
1	imageinput 56x56x3 images with 'zerocenter' normalization	Image Input	56×56×3
2	conv_1 256 3x3x3 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	54×54×256
3	batchnorm_1 Batch normalization with 256 channels	Batch Normalization	54×54×256
4	relu_1 ReLU	ReLU	54×54×256
5	conv_2 256 3x3x256 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	52×52×256
6	batchnorm_2 Batch normalization with 256 channels	Batch Normalization	52×52×256
7	relu_2 ReLU	ReLU	52×52×256
8	maxpool_1 2x2 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	26×26×256
9	conv_3 256 3x3x256 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	24×24×256
10	batchnorm_3 Batch normalization with 256 channels	Batch Normalization	24×24×256
11	relu_3 ReLU	ReLU	24×24×256
12	conv_4 256 3x3x256 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	22×22×256
13	batchnorm_4 Batch normalization with 256 channels	Batch Normalization	22×22×256
14	relu_4 ReLU	ReLU	22×22×256
15	maxpool_2 2x2 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	11×11×256

12	conv_4 256 3x3x256 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	22×22×256
13	batchnorm_4 Batch normalization with 256 channels	Batch Normalization	22×22×256
14	relu_4 ReLU	ReLU	22×22×256
15	maxpool_2 2x2 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	11×11×256
16	conv_5 512 3x3x256 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	9×9×512
17	batchnorm_5 Batch normalization with 512 channels	Batch Normalization	9×9×512
18	relu_5 ReLU	ReLU	9×9×512
19	conv_6 512 3x3x512 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	7×7×512
20	batchnorm_6 Batch normalization with 512 channels	Batch Normalization	7×7×512
21	relu_6 ReLU	ReLU	7×7×512
22	conv_7 512 3x3x512 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	5×5×512
23	batchnorm_7 Batch normalization with 512 channels	Batch Normalization	5×5×512
24	relu_7 ReLU	ReLU	5×5×512
25	maxpool_3 2x2 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	2×2×512
26	fc 4 fully connected layer	Fully Connected	1×1×4
27	softmax softmax	Softmax	1×1×4

Algorithm 1 - Results

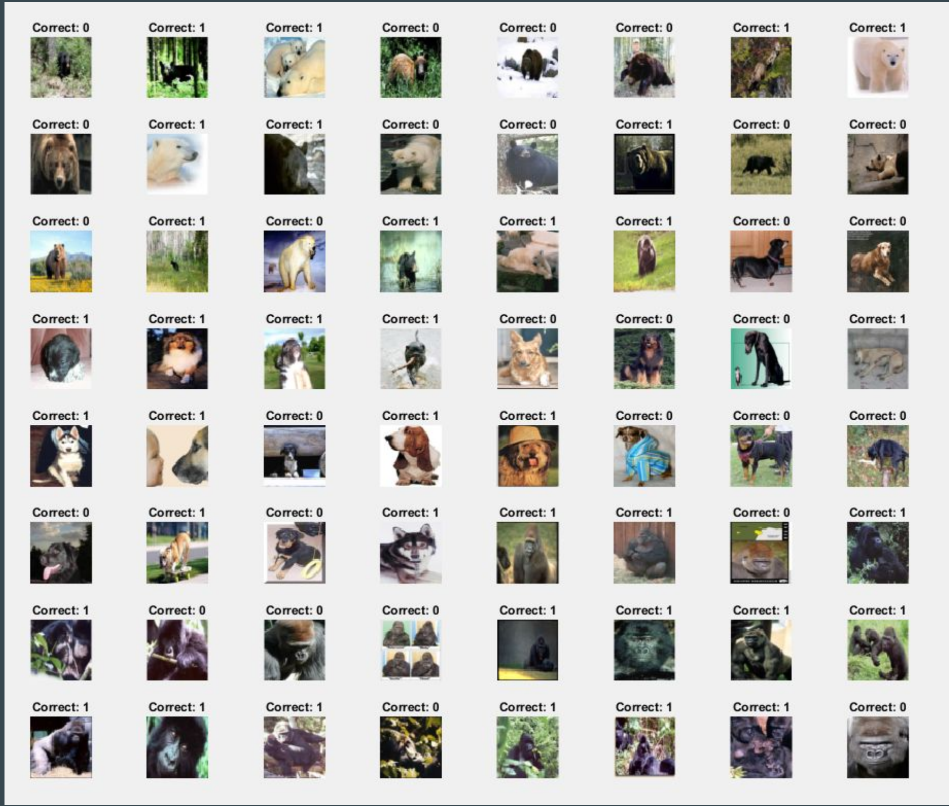


- Validation Accuracy: 79%
- Train Time: 45 sec



Demo - Basic CNN

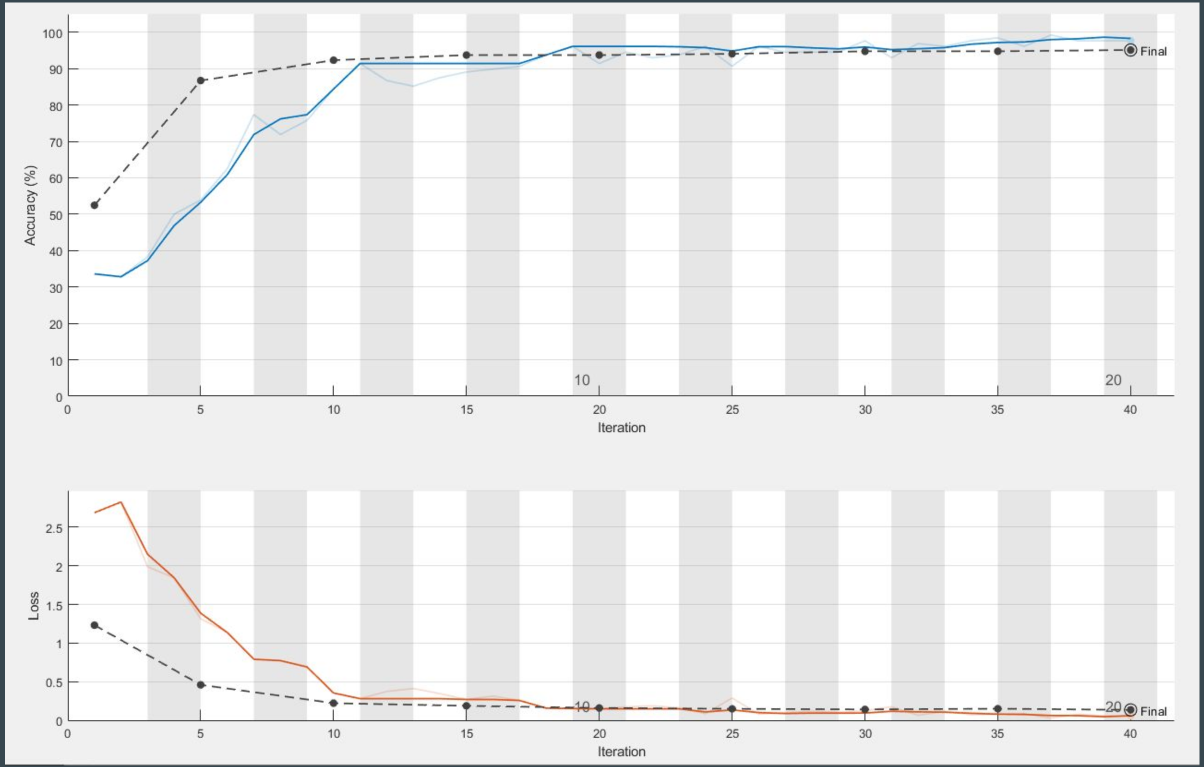
```
%% Load Images
path = '4_AnimalCategories';
imds = imageDatastore(path, 'IncludeSubfolders',true,...
    'LabelSource','foldernames');

%% Pre-process images
aimds = augmentedImageDatastore([56, 56, 3], imdsTrain,...
    'ColorPreprocessing', 'gray2rgb');

%% Read images
img = read(aimds);

%% Use CNN to classify images
pred = predict(net, img(1:64,1));
```



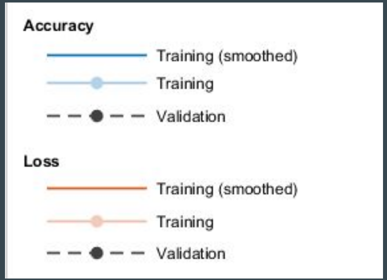
Algorithm 2 - Transfer Learning

- Transfer learning from AlexNet
- Modified final layers to classify our dataset
- 40 iterations, 20 epochs
- 92-95% accuracy consistently

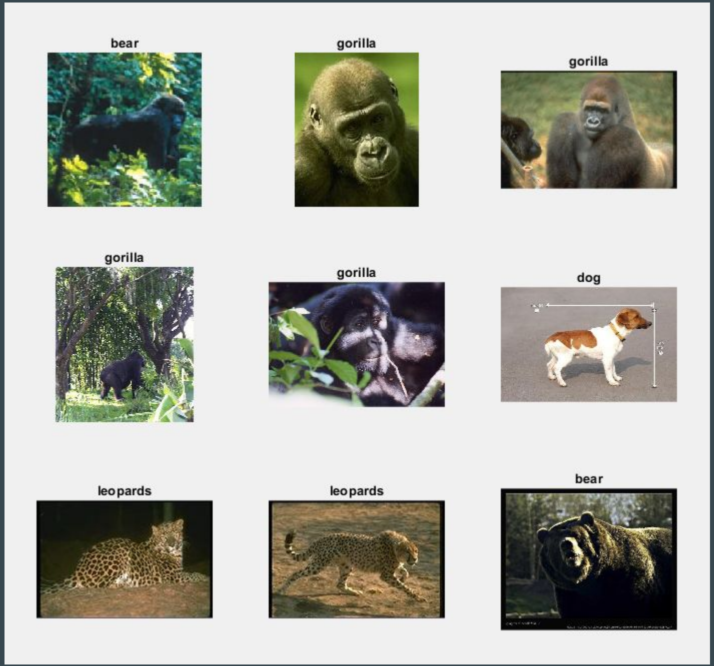
Results - Transfer Learning



- Validation Accuracy: 95.1%
- Elapsed Time: 5 mins 30 secs (running on CPU)



Demo - Transfer Learning



- Still not perfect (8/9 accuracy)
- Incorrectly classified gorilla as bear

Algorithm 3 - CNN training using keras

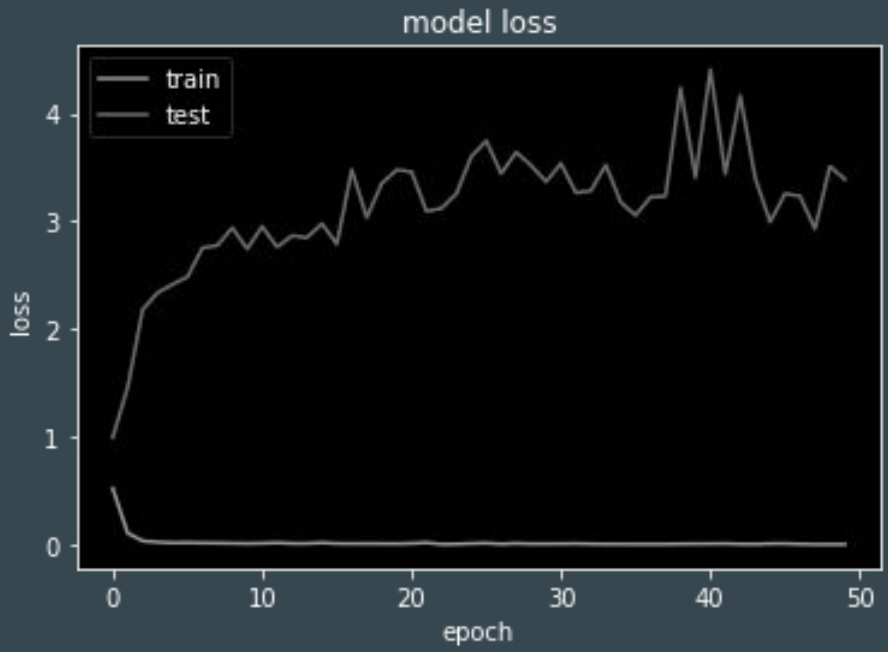
- Regular CNN
- 2 convolution layers
- 1 Full connect layer
- Input shape (64, 64)
- Optimizing using gradient descent

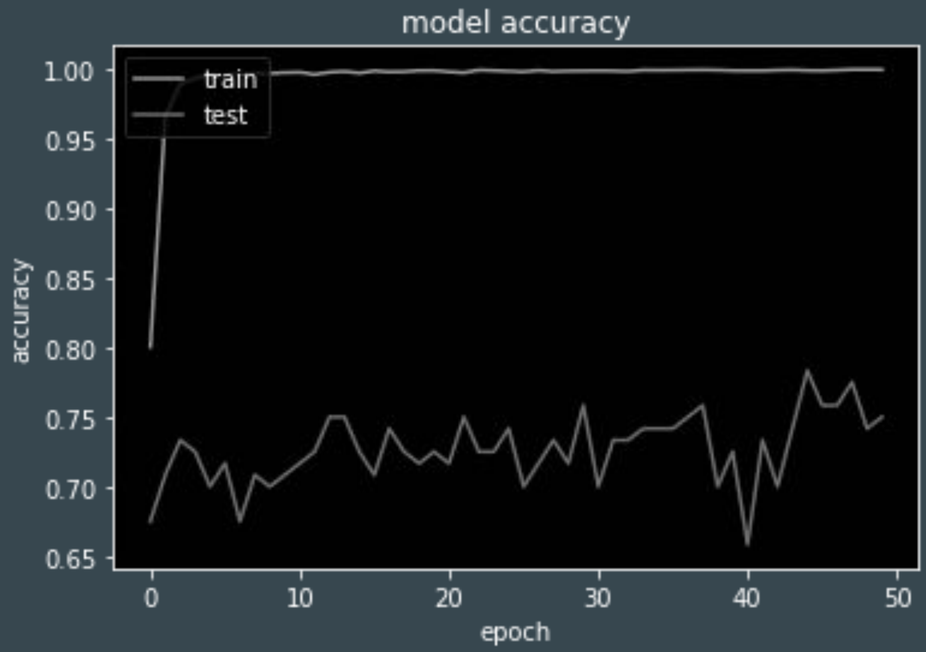
```python
# Initializing the CNN
classifier = Sequential()

# 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
    )
```

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

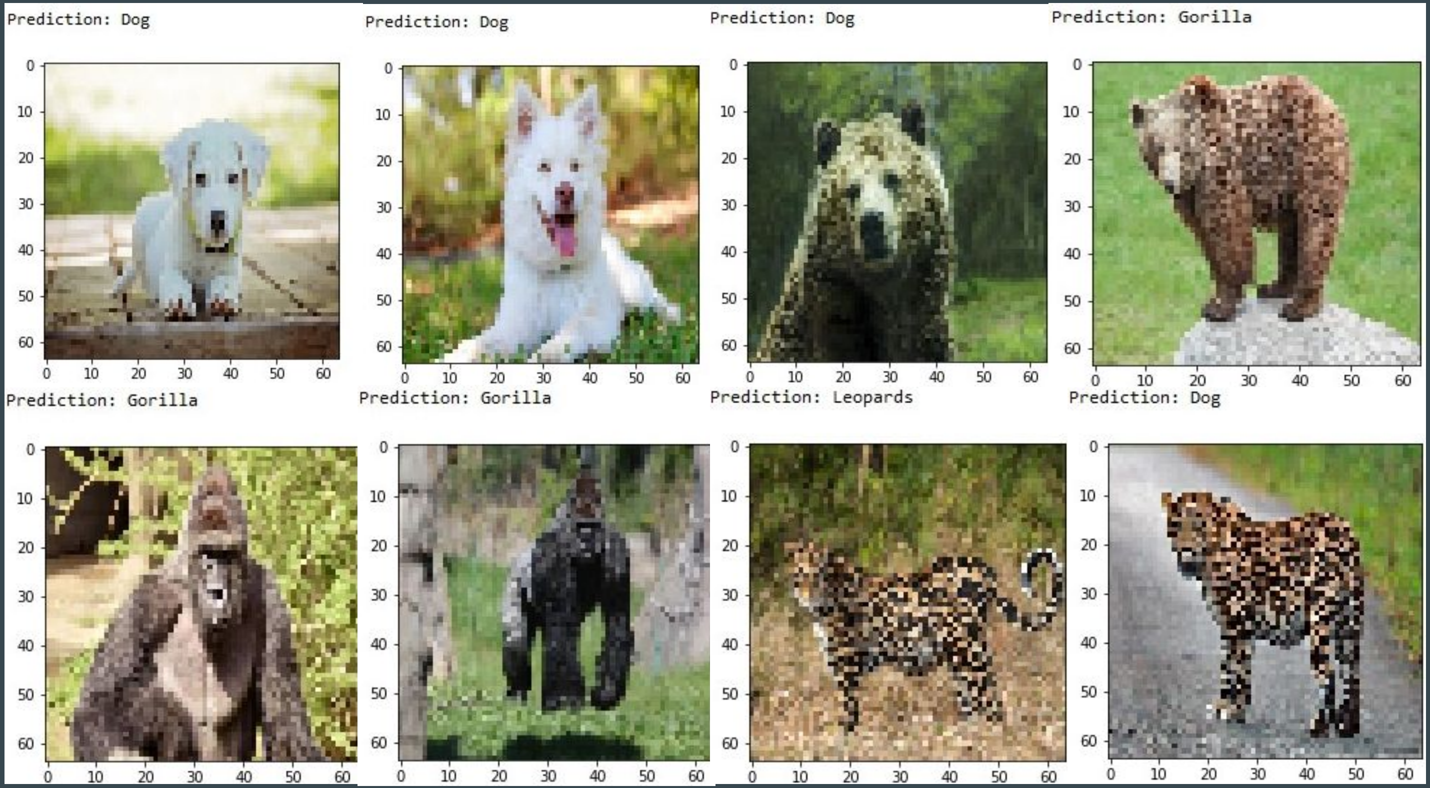
Accuracy and loss

Best Accuracy: 78.33%





Result



Overfitting!!!

Other Approaches

- CNN + SVM
  - Use CNN to extract features
  - Use SVM for classification