COZMO

Object Detection With Anki's AI Robot

Anne Warren

The Origin of the Idea

Computer vision is a hot and growing field of Artificial Intelligence which complexity and worldwide applicability triggered my interest. The global computer vision market reached 11.5 billions in 2020.
The applications:

- Object Detection (cats, cells, tumors, people, guns,…)
- Medical training, robotic surgery,
- Biometric authentication,
- Sport activity tracking,
- Movement analysis,
- Disease progression scoring,
- Self-driving cars,
- Security, video surveillance…etc..

Cozmo is an AI Robot made by Anki that was first commercialized for Christmas 2016. I bought my first Cozmo in 2017, starring here. Anki sold 1.5 millions robots and released a Cozmo SDK compatible with Python. A second generation robot named “Vector” has been released in 2018. Though Anki shut down in 2019, Cozmo enthusiasts continue developing programs, applications and educational curriculums.

The Plan

Cozmo learns to identify an object which holds a sign. The sign is a letter- C, O, Z, M.

The 4 classes of images are labeled and subsequently loaded into a Keras CNN model.

When asked to identify one sign among the four, Cozmo finds the correct sign, says what it is and is happy!

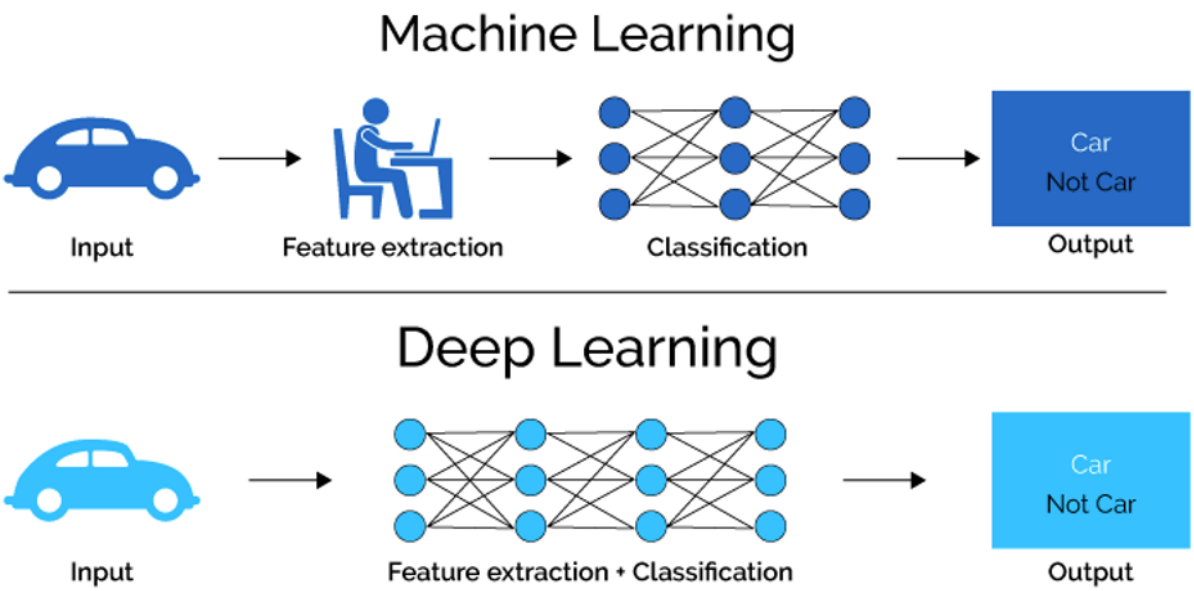
Cozmo SDK

Python scripting is done as usual on the computer using the Cozmo library. The communication between the computer and Cozmo is done by a cell phone in developer mode.
The interface is managed by the Cozmo application in SDK mode.

A viewer allows to see in real-time what Cozmo sees while it is having fun executing the program. Everything Cozmo does via the entertainment side of the application can be performed through the SDK … and even more!

Why Deep Learning?

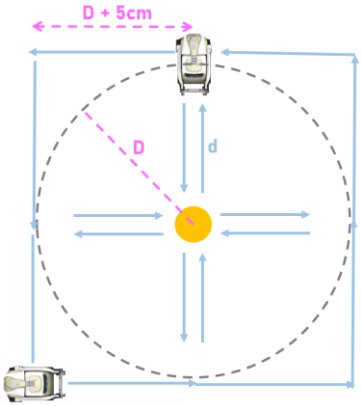
One of the great and practical advantage of Deep Learning over Machine Learning for image classification and object detection is that there is minimal image preparation because the Deep Learning network extracts features and classifies.

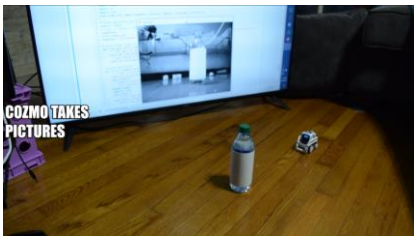


https://ankicozmorobot.com/wp-content/uploads/2018/01/sdk-setup.jpg
https://xd.adobe.com/ideas/principles/emerging-technology/what-is-computer-vision-how-does-it-work/

Sampling Strategy

Cozmo samples images with his video camera The recording needs to be turned on when Cozmo is facing the object and turned off when Cozmo is moving around. Cozmo is about 10cm long and turns around is center point. If Cozmo has to drive a given distance and make a turn, about 5 cm should be added to this distance.





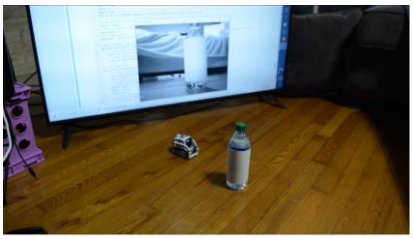
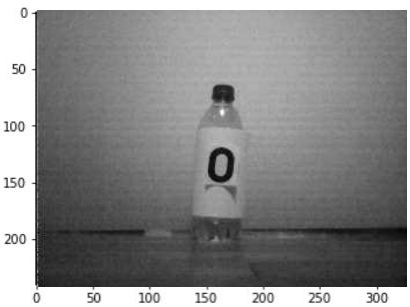


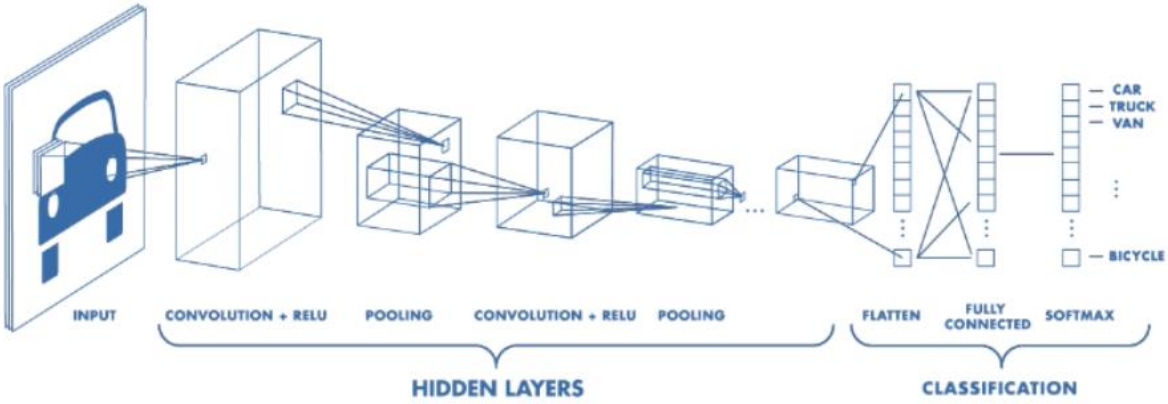
Image Selection and Preparation

The sampling of images was done in a neutral environment to simplify the background and to avoid interferences. Cozmo takes ~150 photos per sampling session. Images are black and white, 320 pixels wide, 240 pixels high with 3 channels. Although the environment is controlled, the generated images have to be reviewed to remove anomalies (e.g. when Cozmo hits the bottle resulting to an extreme close-up). The preparation of the images involved normalization and labelization using Keras preprocessing tools.

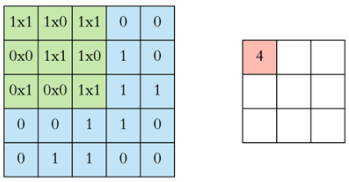


```
train_img = ImageDataGenerator(rescale=1./255,  shear_range=0.2,
                               zoom_range=0.2, horizontal_flip=True)

test_img = ImageDataGenerator(rescale=1./255)

train_generator = train_img.flow_from_directory(
                    path_train_img, target_size=(img_width, img_height),
                    batch_size=batch_size, class_mode='categorical')

validation_generator = test_img.flow_from_directory(
                    path_validation_img, target_size=(img_width, img_height),
                    batch_size=batch_size,  class_mode='categorical')
```

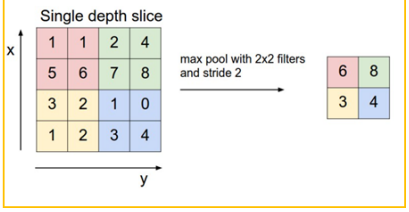
How the CNN Model Works



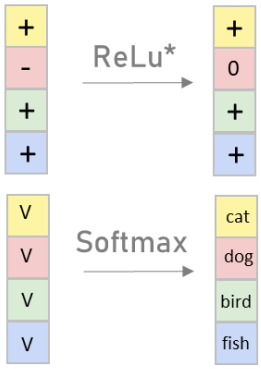
Convolutional Layer



Max Pooling Layer

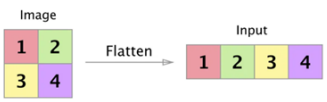


Activations

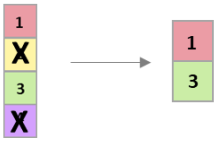


*Rectified Linear Unit

Flattening Layer



Dropout Layer



The CNN model starts with a series of convolution+pooling +activation layers which extract high level features from the image/data while progressively reducing the size of the data. In each layer a filter (here 3x3) slides across the data (here 5x5) and compute the dot products between the filter's entries and the input at each position. The values in the filter are updated each time the network performs backpropagation. The Max Pooling layer reduces further the data by sliding a filter (here 2x2) through the convolutional layer and takes the max value at each position. The Rectified Linear Unit (ReLu) activation let pass positive values and zero out the negative ones. This output is flattened to a 1D array to be fed to fully connected layers for the classification. Dropout layers are used to randomly drop data to promote the non-linearity of the data and improve training time. The softmax activated output layer provides the classes.

https://www.freecodecamp.org/news/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050/
Architecture of a CNN https://www.mathworks.com/videos/introduction-to-deep-learning-what-are-convolutional-neural-1489512765771.html-network

My Keras Model

```
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(240,320,3)))
model.add(Activation('relu'))
```
1st convolutional layer with 32 filters of size 3x3 with ReLu activation

```
model.add(MaxPooling2D(pool_size=(2, 2)))
```
Subsample by taking the max pixel in each 2x2

```
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
```
2nd convolutional layer with 32 filters of size 3x3 with ReLu activation

```
model.add(MaxPooling2D(pool_size=(2, 2)))
```
Subsample by taking the max pixel in each 2x2

```
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
```
3rd convolutional layer with 64 filters of size 3x3 With ReLu activation

```
model.add(MaxPooling2D(pool_size=(2, 2)))
```
Subsample by taking the max pixel in each 2x2

```
model.add(Flatten())
```
Turn 2D data into 1D to feed subsequent layers

```
model.add(Dense(64))
model.add(Activation('relu'))
```
Fully connected layer with 64 filters

```
model.add(Dropout(0.5))
```
50% of the neurons get to contribute

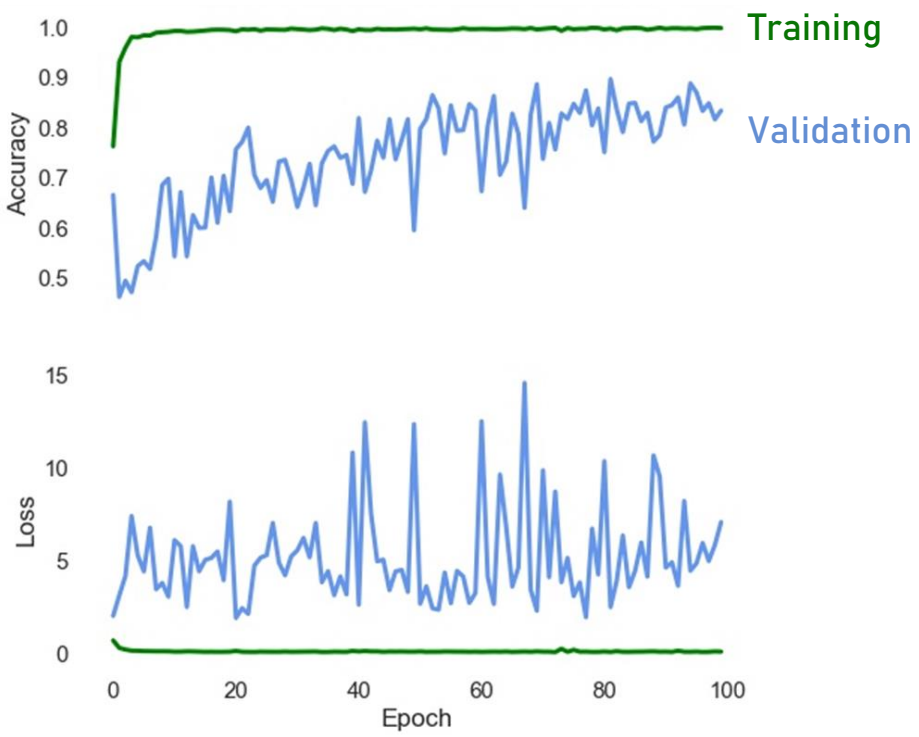
```
model.add(Dense(4))
model.add(Activation('softmax'))
```
Output one out of four classes with the Softmax activation.

Model Performance and Metrics

The model was trained on 4,646 images and validated with 2,795 images. The model is optimized by root mean square propagation (rmsprop) which takes away the need to adjust the learning rate because rmsprop adapts the learning rate during gradient descent as part of its algorithm. Rmsprop uses a moving average of squared gradients to normalize the gradient which balances the step size or momentum (i.e. decrease for large gradients, increase for small gradients). The loss function is Categorical_CrossEntropy as recommended in the Keras documentation for one-hot encoded label, multi-class models. It computes the cross-entropy loss between the labels and predictions. The batch size is 30 to avoid overloading memory. Running the model over 100 epochs takes about 12 hours. The model was tested using “best weight callbacks” and without. The model without call backs reached an accuracy of 83% after 100 epochs while the model with the “ best callbacks” reached an accuracy of 77% after 100 epochs. The accuracy and loss curves suggest that the validation images are not representative and that both training and validation images should be shuffled. Additional images would be beneficial as well.

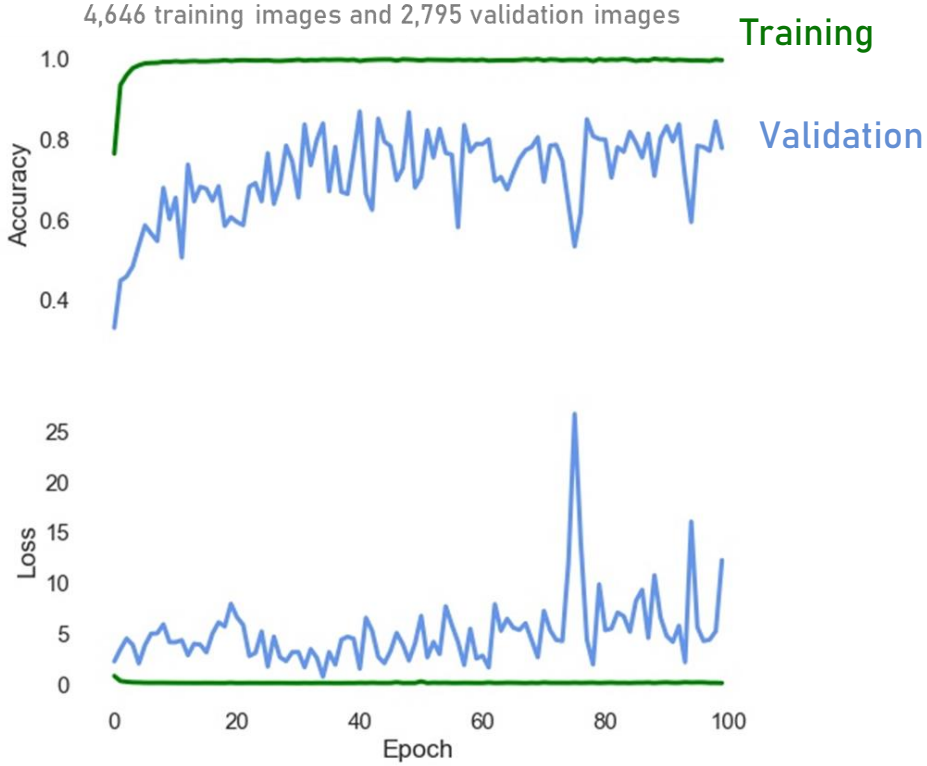
Without Callbacks Accuracy = 83%

loss: categorical_crossentropy , optimizer: rmsprop
4,646 training images and 2,795 validation images



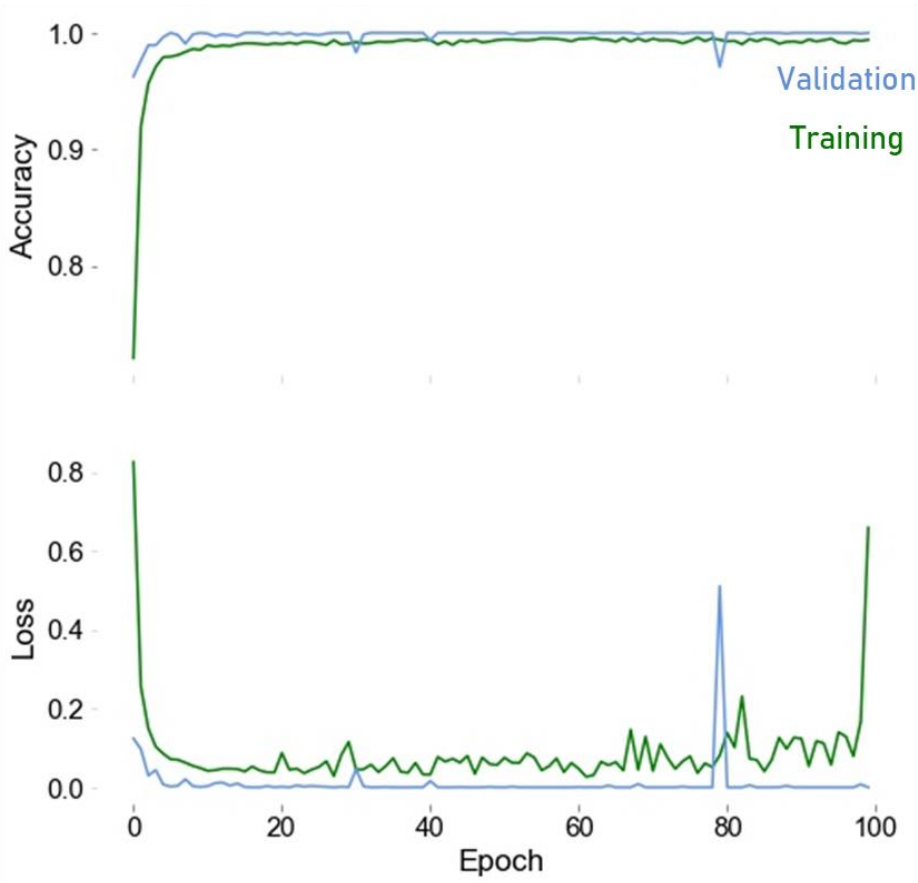
With Callbacks Accuracy = 77%

loss: categorical_crossentropy , optimizer: rmsprop
4,646 training images and 2,795 validation images



After Shuffle Accuracy = 99.8%

loss: categorical_crossentropy , optimizer: rmsprop
7,352 training images and 1,600 validation images



Shuffling the image deck made a huge difference. The accuracy moved up to 99.8%.
The validation curve (blue) shows higher accuracy than the training curve (green) because the model has a dropout layer (50% drop).
The model reaches optimum predictive capability after 10 epochs.
Interestingly both curves show that the model is running into issues around 30 and 80 epochs.

Model Application

Cozmo looks at an item and collects new images. The images are normalized and sent to the model. The model classifies the images and provides an answer.
Cozmo says what it is and giggles when he identifies a known letter.

"This is the letter C like Cozmo"
" I don't know this one"

Conclusion and Future Work

Simple Convolutional Neural Networks are relatively easy to build and make very powerful models for image classification and object detection.

The sequential model used in this project has good accuracy (>80%) but the accuracy curve shows that there is a need to shuffle the images between training and validation. After shuffling he model reaches an accuracy of 99.8%.

As of now Cozmo just says the prediction of the model or "I don't know this one" if the class is not known from the model yet. A confidence level using the model uncertainty and data uncertainty could be added to the program to prevent Cozmo from saying a wrong prediction.

New features will be added:

- new classes (the whole alphabet),
- Cozmo will read words by putting together the letters it sees,
- Cozmo will wander around and be triggered by an object/letter it recognizes.