Image Classification with CNN: Cats vs Dogs

•••

Jack D

Disclaimer

This code was developed via a guide published by Jay ‘Coding Lane’ Patel on YouTube.

The following slides regard the explanation of how CNN’s work and the python implementation of the neural network.

Intro to Data

- Kaggle’s Cats and Dogs dataset
  - Not all is used (a lot of data)
  - Original Dataset has approx 12,000 pictures
    - We will only use 2,400
  - https://www.microsoft.com/en-us/download/details.aspx?id=54765
- Data transformed externally from JPG to CSV (containing RGB data)
- Each image is 100*100 pixels (each containing 3 rgb values)

ex:

ex:

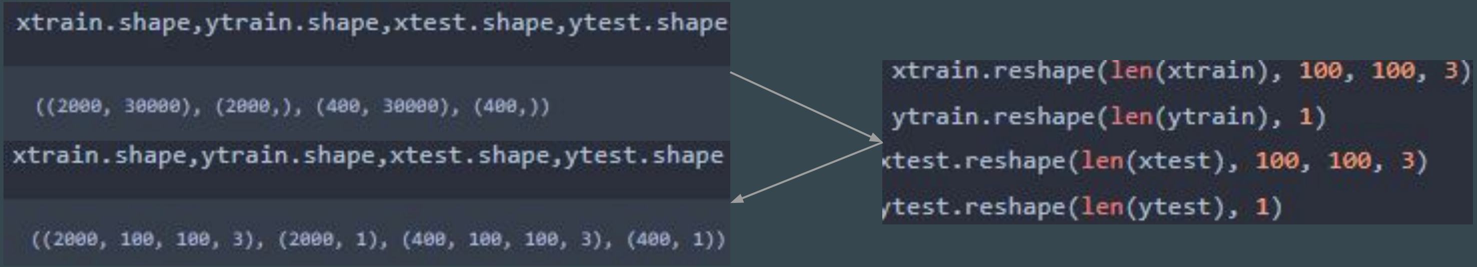
ex:

ex:

ex:

Preprocessing

- Original dataset shape is (2,400 by 30,000)
- Needed to reshape to (2,400 tuples of 100*100*3(rgb))

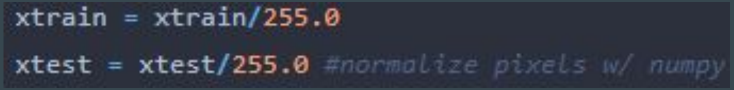


```
xtrain.shape,ytrain.shape,xtest.shape,ytest.shape

((2000, 30000), (2000,), (400, 30000), (400,))
```

```
xtrain.reshape(len(xtrain), 100, 100, 3)
ytrain.reshape(len(ytrain), 1)
xtest.reshape(len(xtest), 100, 100, 3)
ytest.reshape(len(ytest), 1)
```

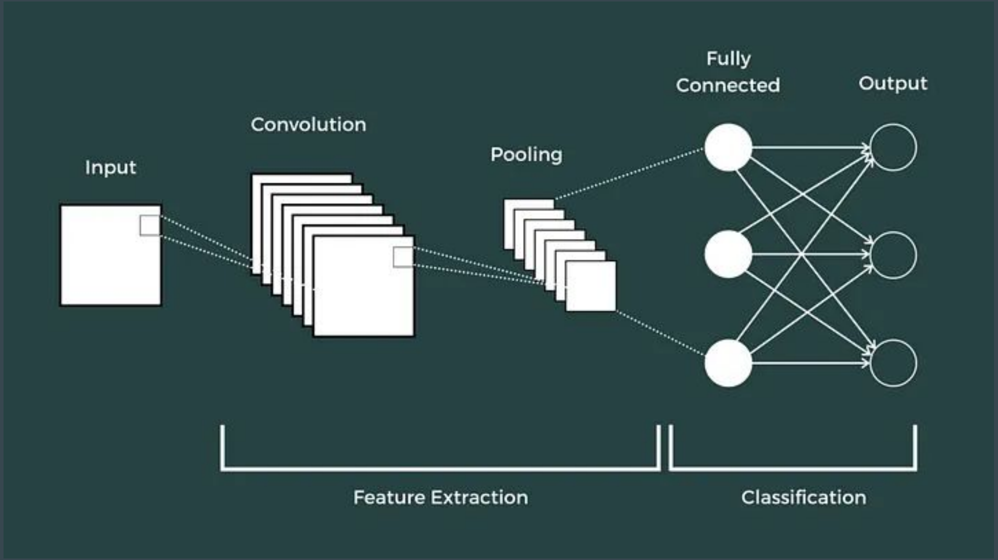
```
xtrain.shape,ytrain.shape,xtest.shape,ytest.shape

((2000, 100, 100, 3), (2000, 1), (400, 100, 100, 3), (400, 1))
```

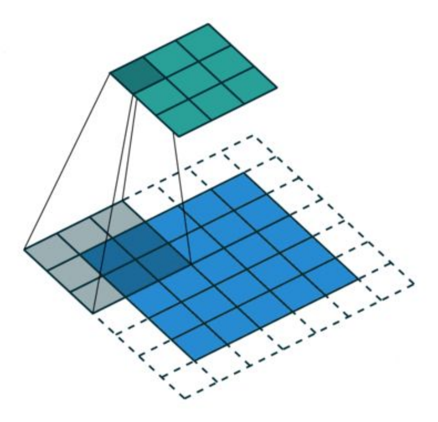
- Need to rescale RGB values from 0-255 to 0-1 for input

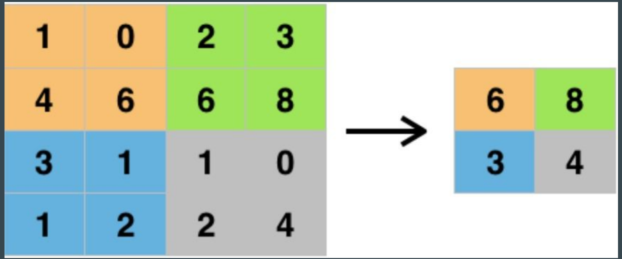


```
xtrain = xtrain/255.0
xtest = xtest/255.0 #normalize pixels w/ numpy
```

Intro: Convolutional Neural Networks (CNN)







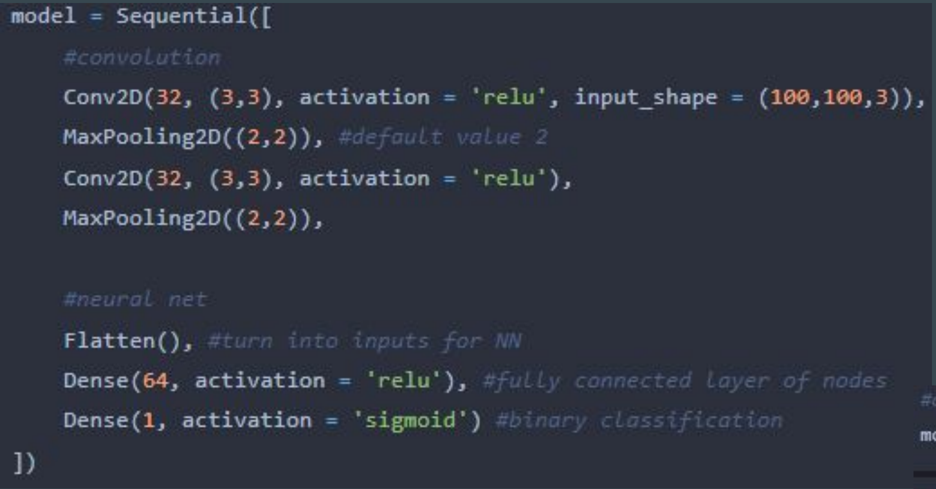
1	0	2	3
4	6	6	8
3	1	1	0
1	2	2	4

→

6	8
3	4

- Convolution - aggregates partitions of original input
- Pooling - amplifies features of image to increase distinctiveness
  - These, and some ReLU nodes, make up our ‘hidden layer’ in the network

Python Implementation of CNN



```
model = Sequential([
    #convolution
    Conv2D(32, (3,3), activation = 'relu', input_shape = (100,100,3)),
    MaxPooling2D((2,2)), #default value 2
    Conv2D(32, (3,3), activation = 'relu'),
    MaxPooling2D((2,2)),

    #neural net
    Flatten(), #turn into inputs for NN
    Dense(64, activation = 'relu'), #fully connected layer of nodes
    Dense(1, activation = 'sigmoid') #binary classification
])
```

Model:

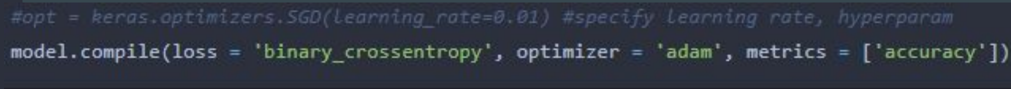
-Convolution of Image = Conv2D (32 filters, 3x3), uses ReLU (ie 0 for activation)

-Pooling = MaxPooling (2x2, steps 2)

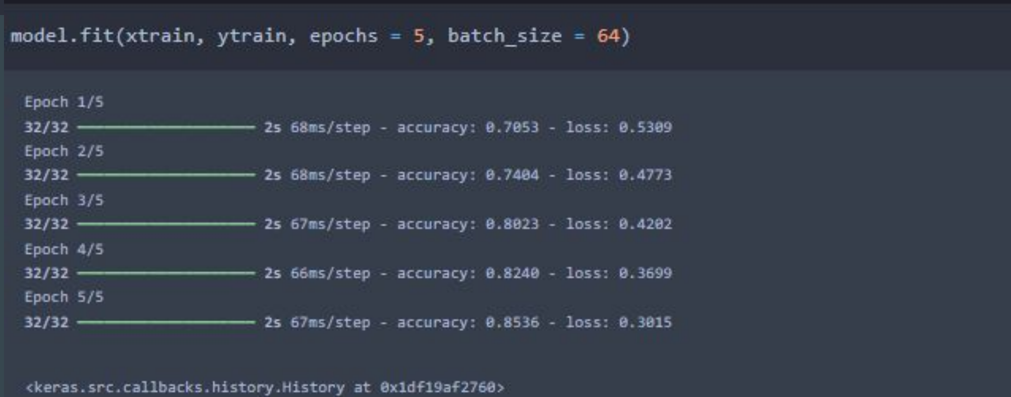
-Flatten = convert image to interpretable array

-Dense = make layer of nodes in NN which will connect to all previous and consecutive nodes

Compiler:
-Minimizes binary entropy.
-Uses adam optimizer (faster training
-than SGD, developed by OpenAI).
-Assesses Accuracy



```
#opt = keras.optimizers.SGD(learning_rate=0.01) #specify learning rate, hyperparam
model.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
```

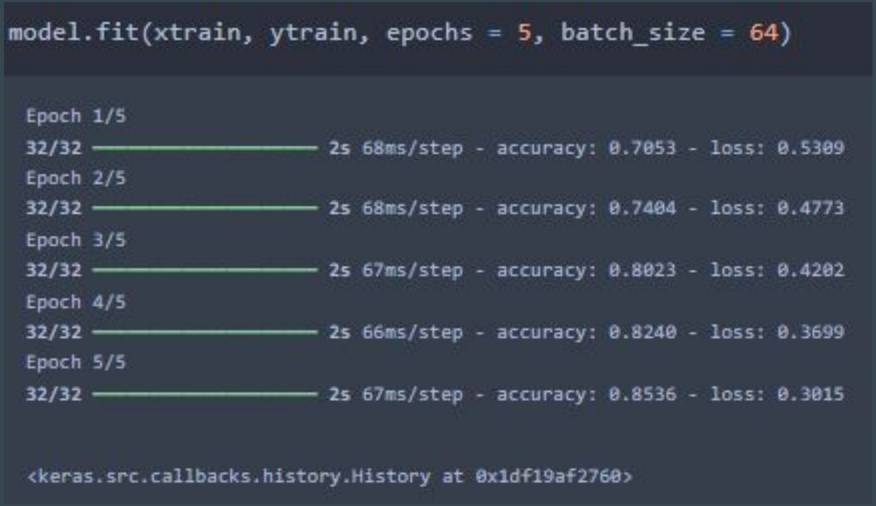


```
model.fit(xtrain, ytrain, epochs = 5, batch_size = 64)
```

```
Epoch 1/5
32/32 ━━━━━━━━━━━━━━━━━━━━ 2s 68ms/step - accuracy: 0.7053 - loss: 0.5309
Epoch 2/5
32/32 ━━━━━━━━━━━━━━━━━━━━ 2s 68ms/step - accuracy: 0.7404 - loss: 0.4773
Epoch 3/5
32/32 ━━━━━━━━━━━━━━━━━━━━ 2s 67ms/step - accuracy: 0.8023 - loss: 0.4202
Epoch 4/5
32/32 ━━━━━━━━━━━━━━━━━━━━ 2s 66ms/step - accuracy: 0.8240 - loss: 0.3699
Epoch 5/5
32/32 ━━━━━━━━━━━━━━━━━━━━ 2s 67ms/step - accuracy: 0.8536 - loss: 0.3015

<keras.src.callbacks.history.History at 0x1df19af2760>
```

Python Implementation of CNN

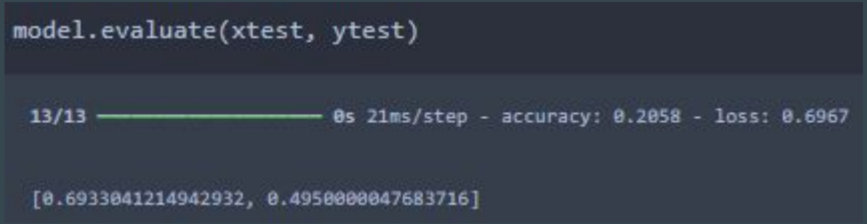
Training of Model and Evaluation



```
model.fit(xtrain, ytrain, epochs = 5, batch_size = 64)

Epoch 1/5
32/32 ━━━━━━━━━━━━━━━━━━━━ 2s 68ms/step - accuracy: 0.7053 - loss: 0.5309
Epoch 2/5
32/32 ━━━━━━━━━━━━━━━━━━━━ 2s 68ms/step - accuracy: 0.7404 - loss: 0.4773
Epoch 3/5
32/32 ━━━━━━━━━━━━━━━━━━━━ 2s 67ms/step - accuracy: 0.8023 - loss: 0.4202
Epoch 4/5
32/32 ━━━━━━━━━━━━━━━━━━━━ 2s 66ms/step - accuracy: 0.8240 - loss: 0.3699
Epoch 5/5
32/32 ━━━━━━━━━━━━━━━━━━━━ 2s 67ms/step - accuracy: 0.8536 - loss: 0.3015

<keras.src.callbacks.history.History at 0x1df19af2760>
```

Hyperparameters Examined

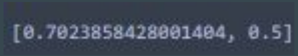
- Different numbers of epochs (training iterations)
- Stochastic Gradient Descent vs. Adam optimization



```
model.evaluate(xtest, ytest)

13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - accuracy: 0.2058 - loss: 0.6967

[0.6933041214942932, 0.4950000047683716]
```

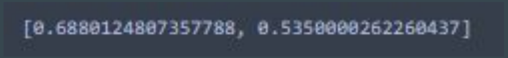
Model Evaluation - SGD(a=0.001)

- Epoch = 1
  - [loss, acc]



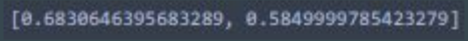
```
[0.7023858428001404, 0.5]
```

- Epoch = 10

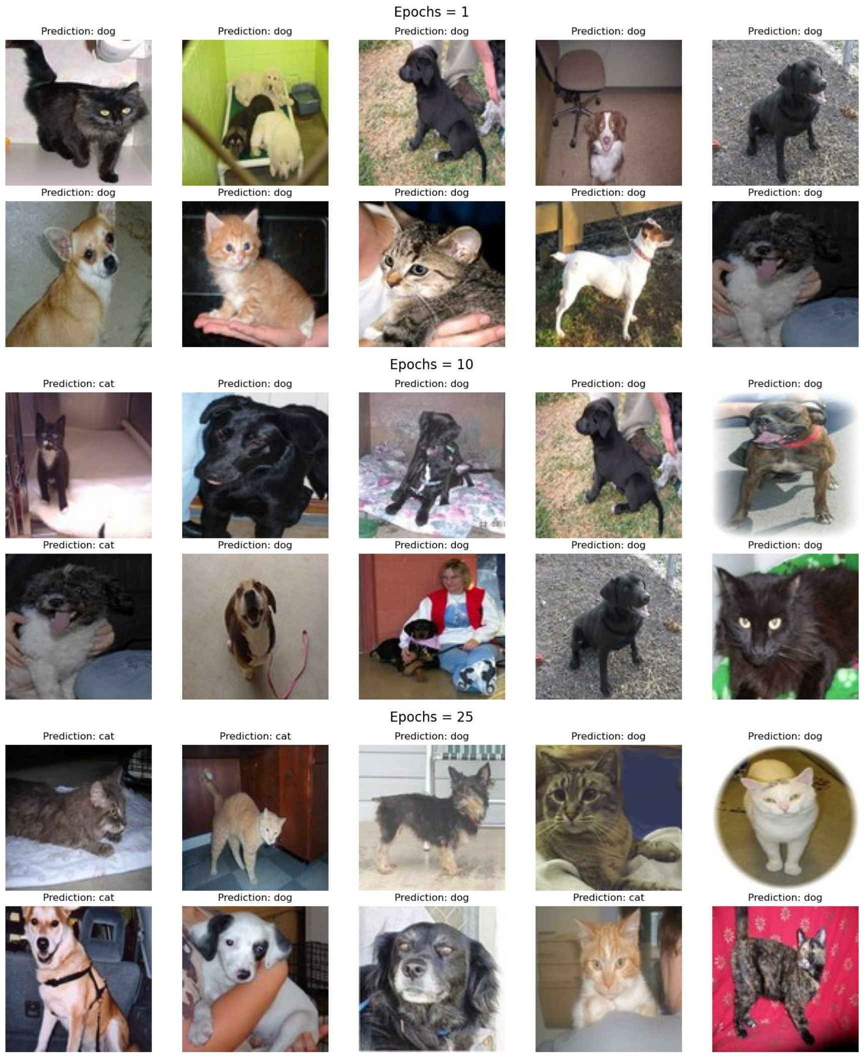


```
[0.6880124807357788, 0.5350000262260437]
```

- Epoch = 25

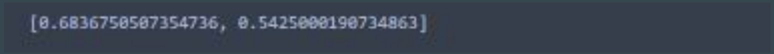


```
[0.6830646395683289, 0.5849999785423279]
```



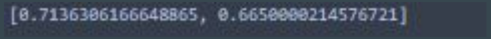
Model Evaluation - Adam

- Epoch = 1
    - [loss, acc]



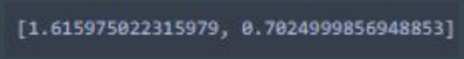
```
[0.6836750507354736, 0.5425000190734863]
```

- Epoch = 10

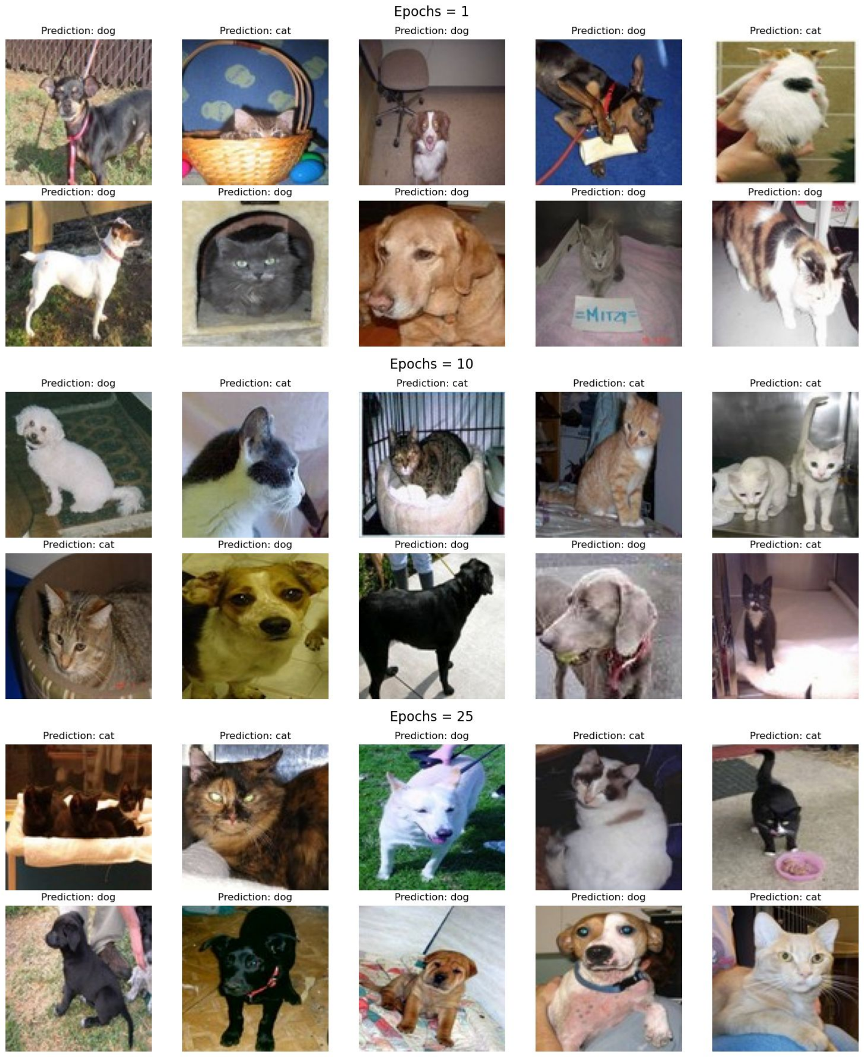


```
[0.7136306166648865, 0.6650000214576721]
```

- Epoch = 25



```
[1.615975022315979, 0.7024999856948853]
```



Conclusion

- The more training iterations (epochs) for a neural network, the higher the accuracy
- In the context of CNN, Adam optimizers were able to utilize epochs more than SGD
- CNN plays a fundamental role in computer vision