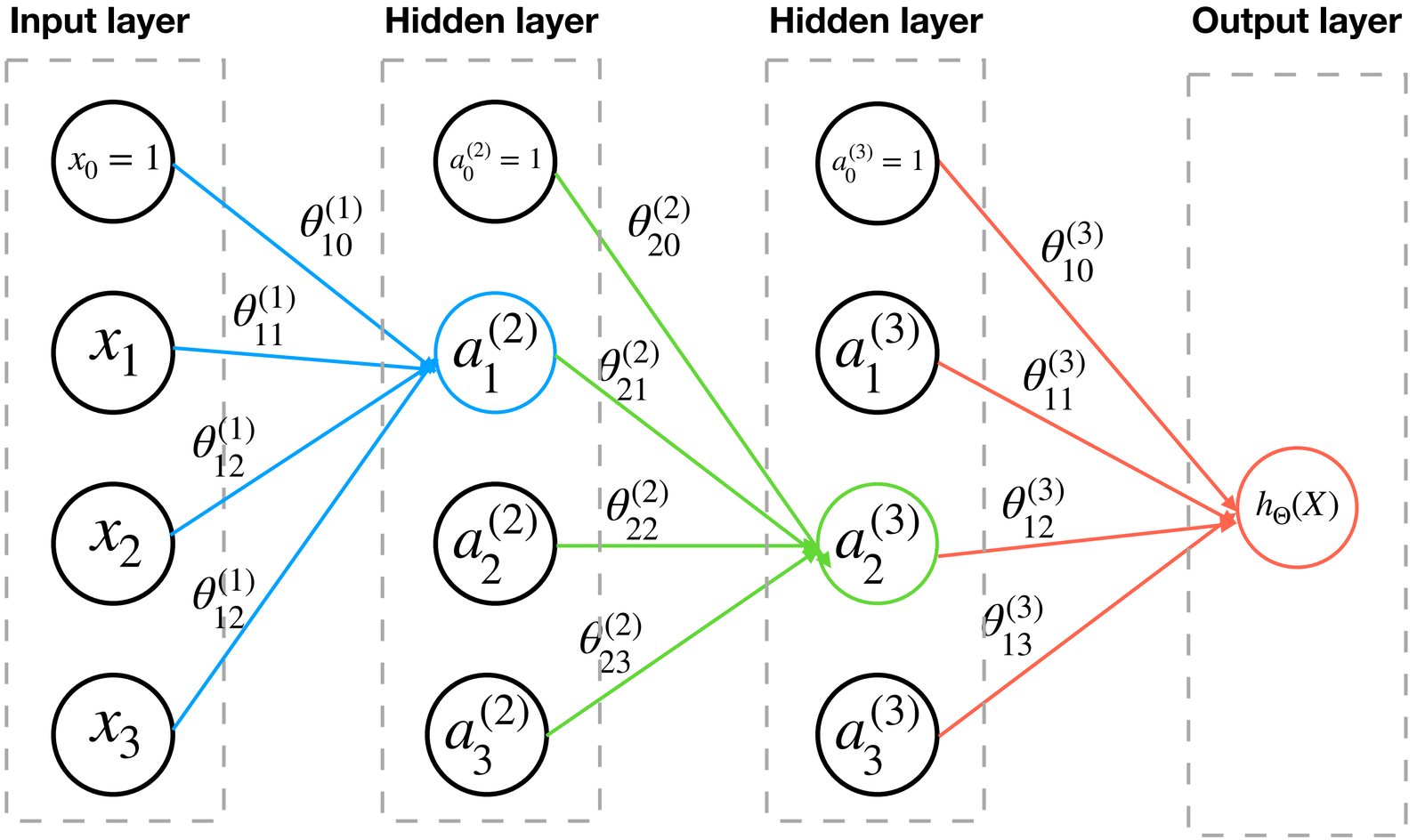
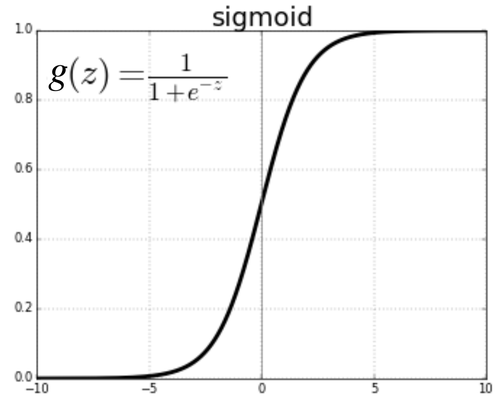
Convolutional Neural Networks

Convolutional Neural Networks

https://github.com/JBCA-MachineLearning/Workshop/tree/master/CNN



$$X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$



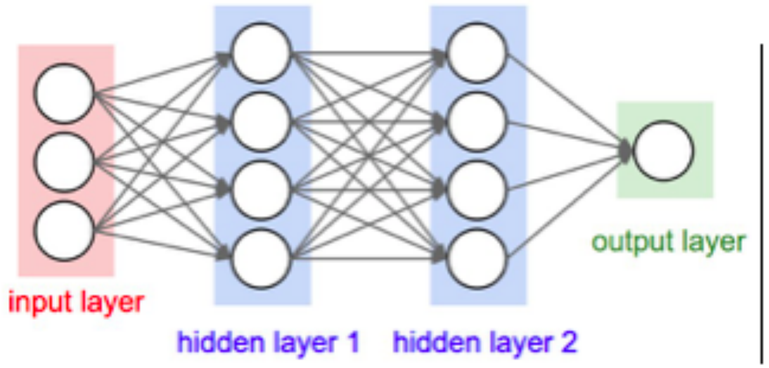
$$a_1^{(2)} = g\left(\theta_{10}^{(1)}x_0 + \theta_{11}^{(1)}x_1 + \theta_{12}^{(1)}x_2 + \theta_{13}^{(1)}x_3\right) = g\left(z_1^{(2)}\right)$$

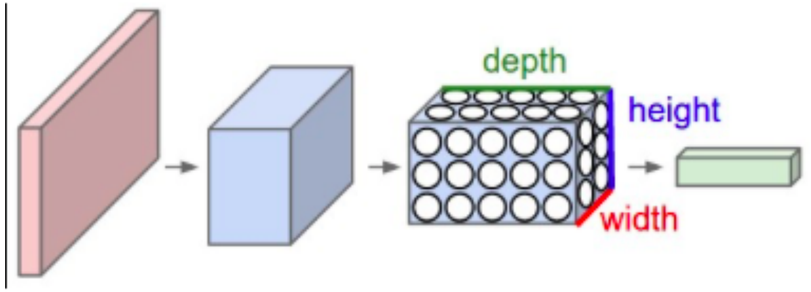
$$a_2^{(3)} = g\left(\theta_{20}^{(2)}a_0^{(2)} + \theta_{21}^{(2)}a_1^{(2)} + \theta_{22}^{(2)}a_2^{(2)} + \theta_{23}^{(2)}a_3^{(2)}\right) = g\left(z_2^{(3)}\right)$$

$$h_\Theta(X) = g\left(\theta_{10}^{(3)}a_0^{(3)} + \theta_{11}^{(3)}a_1^{(3)} + \theta_{12}^{(3)}a_2^{(3)} + \theta_{13}^{(3)}a_3^{(3)}\right) = g\left(z^{(4)}\right)$$

Convolutional layer

- Convolutional layer
- ReLu Layer
- Pooling Layer
- Dropout Layer
- Fully Connected Layer





Convolution Layer

In [ ]:
```
from keras.models import Sequential
from keras.layers import Conv2D

model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=input_shape))
```

Input

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Filter/Kernel

1	0	1
0	1	0
1	0	1

1x1	1x0	1x1	0	0
0x0	1x1	1x0	1	0
0x1	0x0	1x1	1	1
0	0	1	1	0
0	1	1	0	0

4		

Source: https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

Convolution Layer

In [ ]:
```
from keras.models import Sequential
from keras.layers import Conv2D

model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=input_shape))
```

Input

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Filter/Kernel

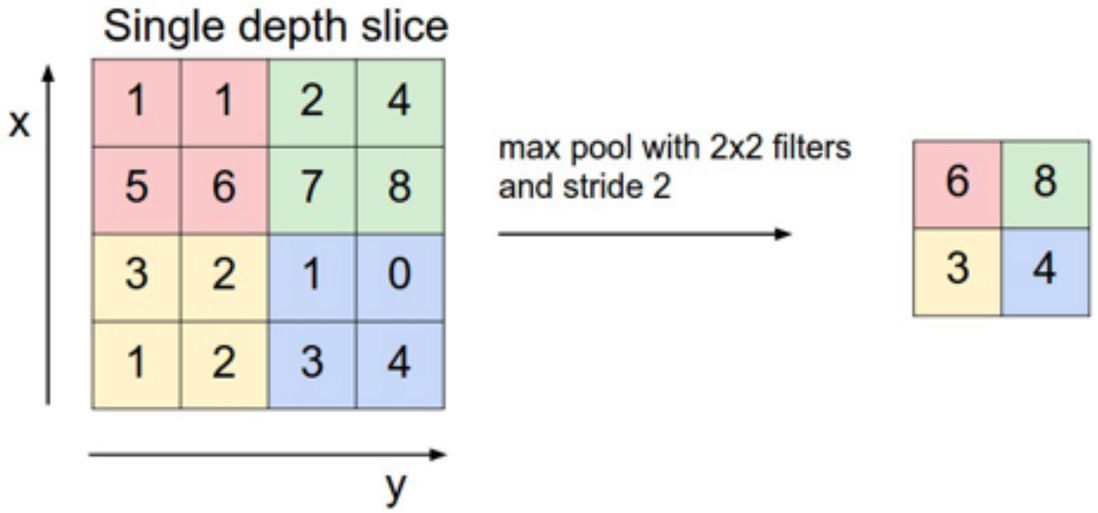
1	0	1
0	1	0
1	0	1

1x1	1x0	1x1	0	0
0x0	1x1	1x0	1	0
0x1	0x0	1x1	1	1
0	0	1	1	0
0	1	1	0	0

4		

Source: https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

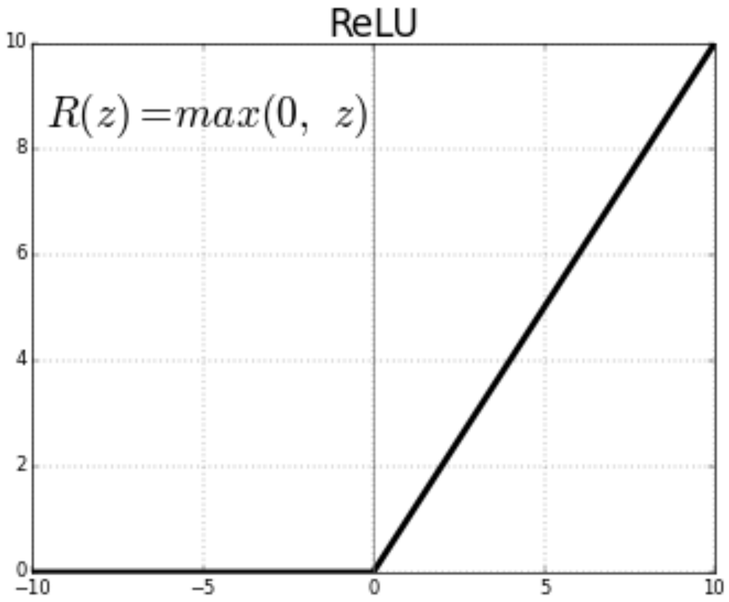
Pooling layer



- **Reduce Image size and therefore number of parameters to be learned**
- **Can help with overfitting**

In [ ]:
```
from keras.layers import MaxPooling2D
model.add(MaxPooling2D(pool_size=(2, 2)))
```

ReLu layer

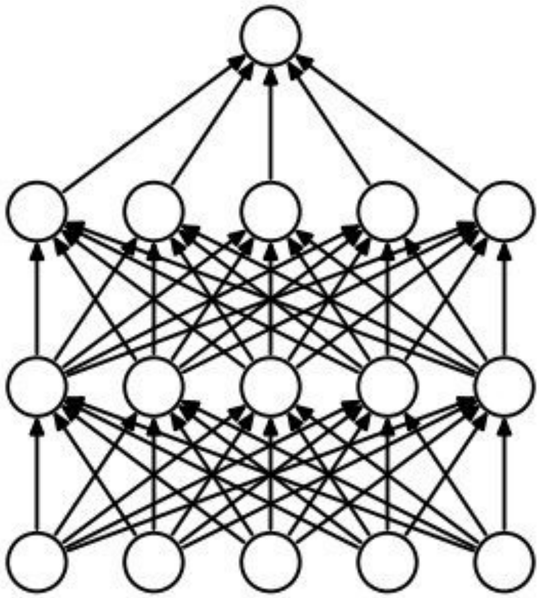


- **Added after convolutional layer**
- **Introduces non-linearity to network**

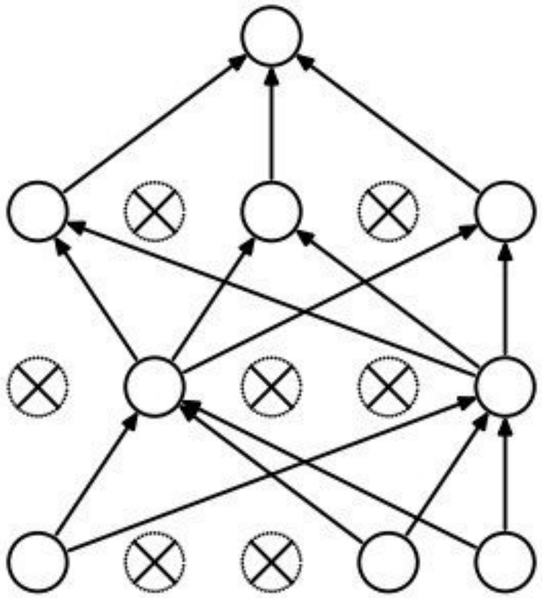
In [ ]:
```
from keras.layers import Activation
model.add(Activation('relu'))
```

Drop out layer

In [ ]:
```
from keras.layers import Activation, Dropout
model.add(Dropout(0.5))
```



(a) Standard Neural Net



(b) After applying dropout.

Putting it all together

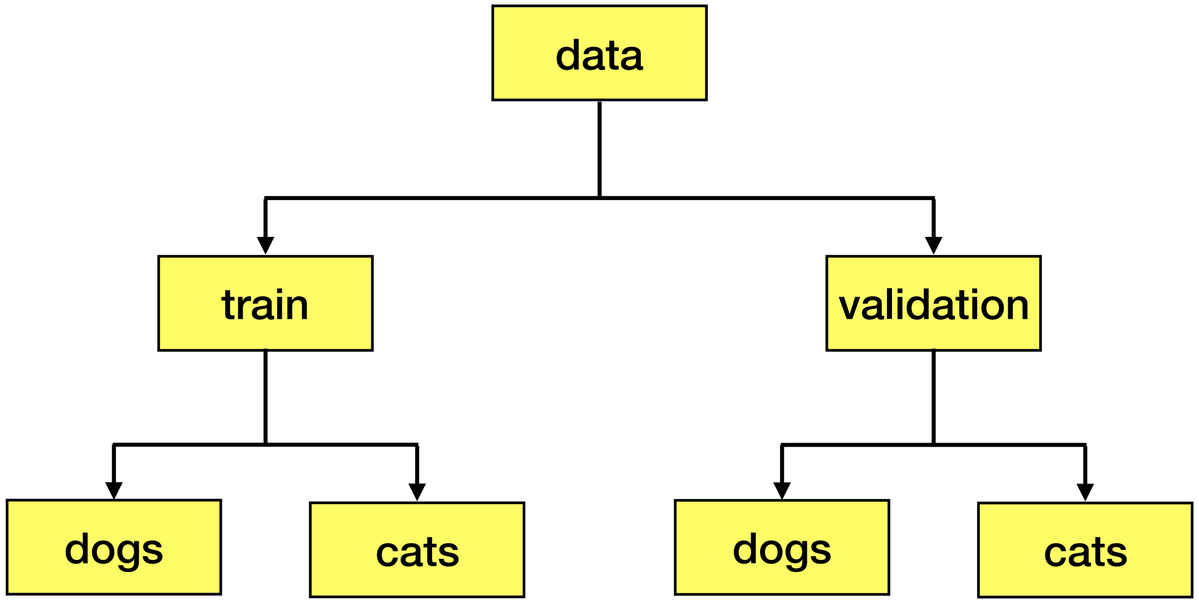
In [2]:

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense

model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=input_shape))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation('sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='rmsprop', metrics=['accuracy'
])
```

```
Using TensorFlow backend.
```

http://scs.ryerson.ca/~aharley/vis/conv/flat.html

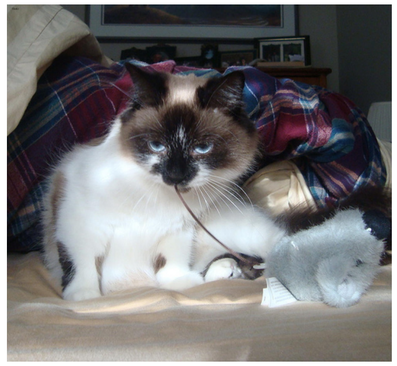
Training Data



http://www.robots.ox.ac.uk/~vgg/data/pets/

Augmenting Data

In [ ]:
```
from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_arr
ay, load_img

datagen = ImageDataGenerator(
        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest')
```



In [69]:
```
from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_arr
ay, load_img

train_data_dir = 'data/train'
validation_data_dir = 'data/validation'
nb_train_samples = 2000
nb_validation_samples = 800
batch_size = 16

# this is the augmentation configuration we will use for training
train_datagen = ImageDataGenerator(
    rescale=1. / 255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)

# this is the augmentation configuration we will use for testing:
# only rescaling
validation_datagen = ImageDataGenerator(rescale=1. / 255)
```

```
Found 2000 images belonging to 2 classes.
Found 800 images belonging to 2 classes.
```

In [ ]:
```
train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='binary')

validation_generator = validation_datagen.flow_from_directory(
    validation_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='binary')
```

Training

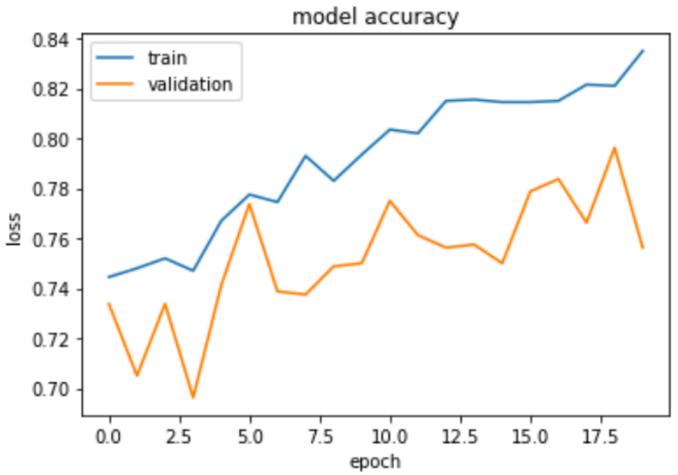
In [20]:

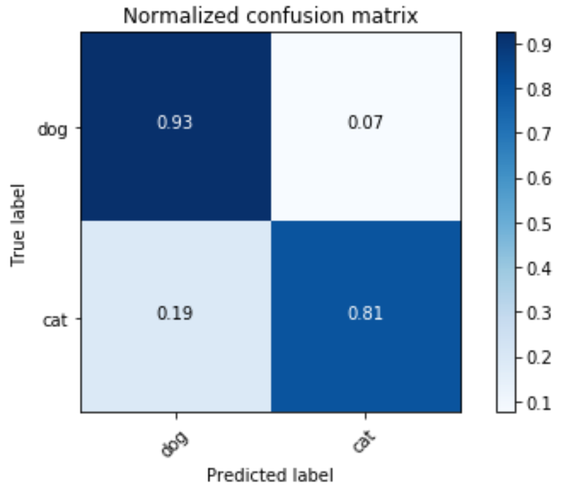
```
history = model.fit_generator(
    train_generator,
    steps_per_epoch=nb_train_samples // batch_size,
    epochs=20,
    validation_data=validation_generator,
    validation_steps=nb_validation_samples // batch_size)

model.save_weights('first_try.h5')
```

```
Epoch 1/20
125/125 [==============================] - 60s 481ms/step - loss: 0.5441 - acc
: 0.7445 - val_loss: 0.5389 - val_acc: 0.7338
Epoch 2/20
125/125 [==============================] - 59s 473ms/step - loss: 0.5246 - acc
: 0.7480 - val_loss: 0.5549 - val_acc: 0.7050
Epoch 3/20
125/125 [==============================] - 63s 504ms/step - loss: 0.5101 - acc
: 0.7520 - val_loss: 0.5298 - val_acc: 0.7338
Epoch 4/20
125/125 [==============================] - 66s 530ms/step - loss: 0.5276 - acc
: 0.7470 - val_loss: 0.5853 - val_acc: 0.6963
Epoch 5/20
125/125 [==============================] - 78s 625ms/step - loss: 0.5004 - acc
: 0.7670 - val_loss: 0.5165 - val_acc: 0.7412
Epoch 6/20
125/125 [==============================] - 73s 586ms/step - loss: 0.4802 - acc
: 0.7775 - val_loss: 0.4903 - val_acc: 0.7738
```

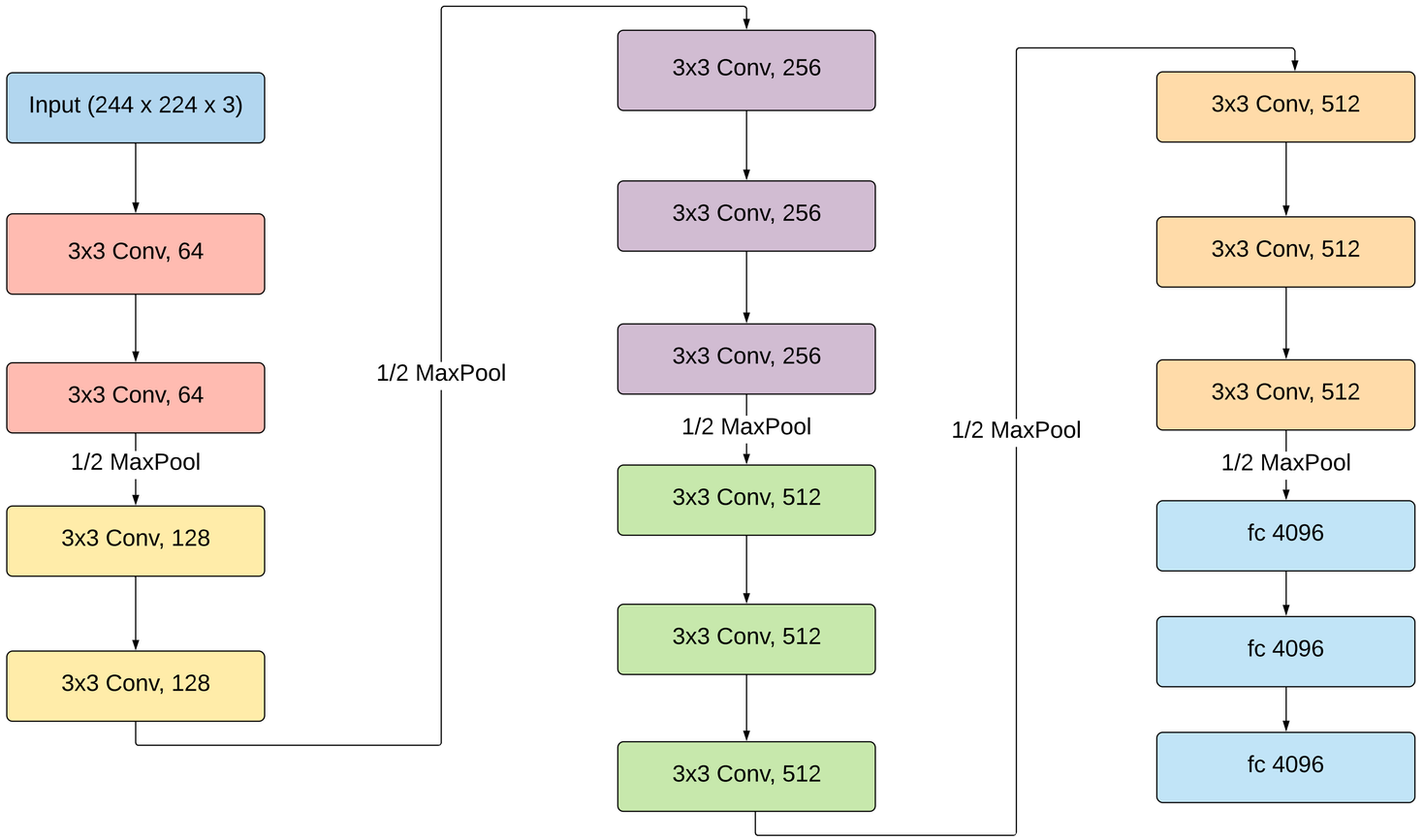
Evaluation

In [22]:
```
import matplotlib.pyplot as plt

plt.plot(history.history['acc'], label='train')
plt.plot(history.history['val_acc'], label = 'validation')
plt.title('model accuracy')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend(loc='upper left')
plt.show()
```





VGG16



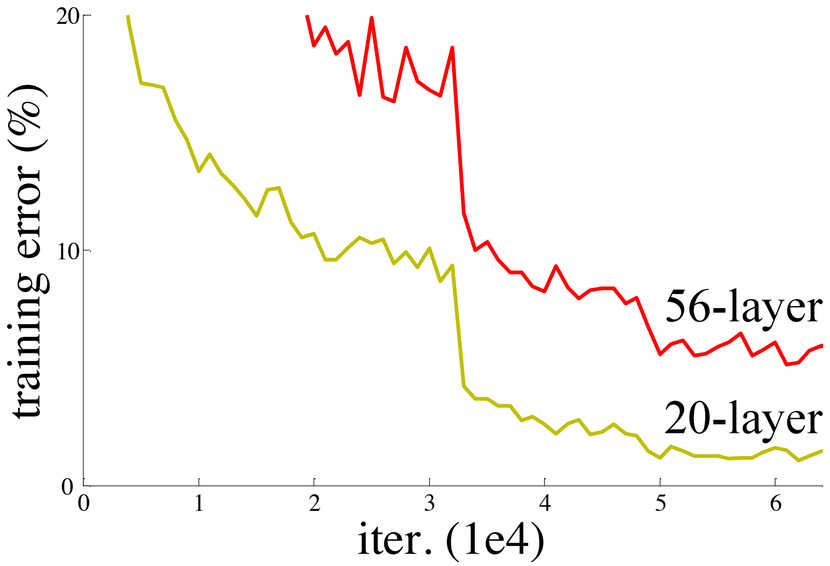
https://arxiv.org/pdf/1409.1556.pdf

ResNet

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

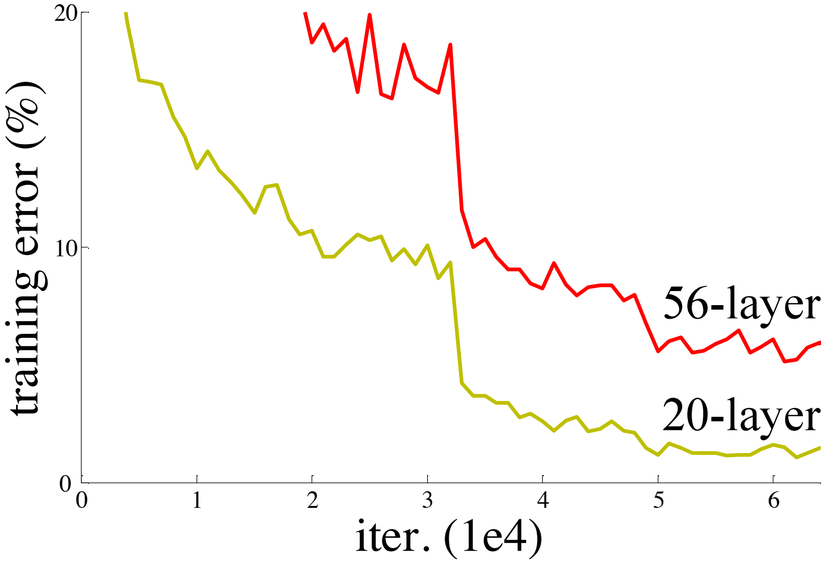
ResNet

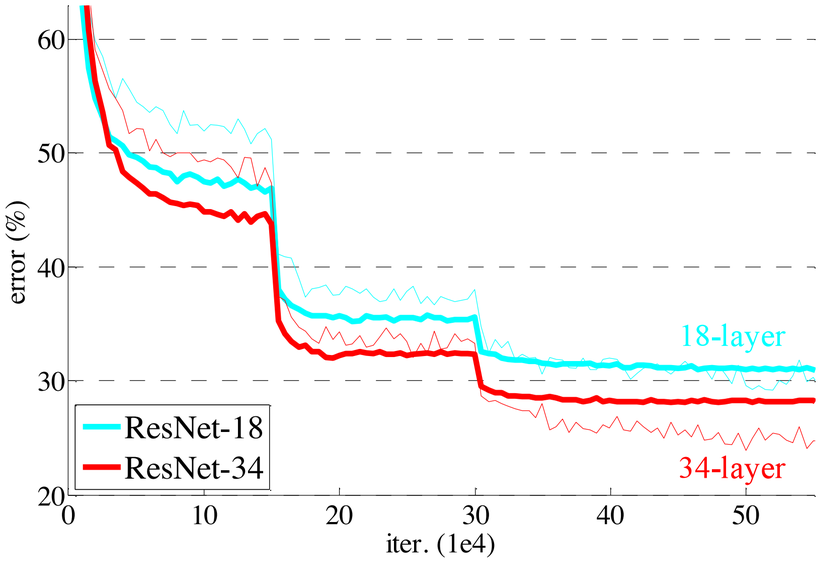
layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9



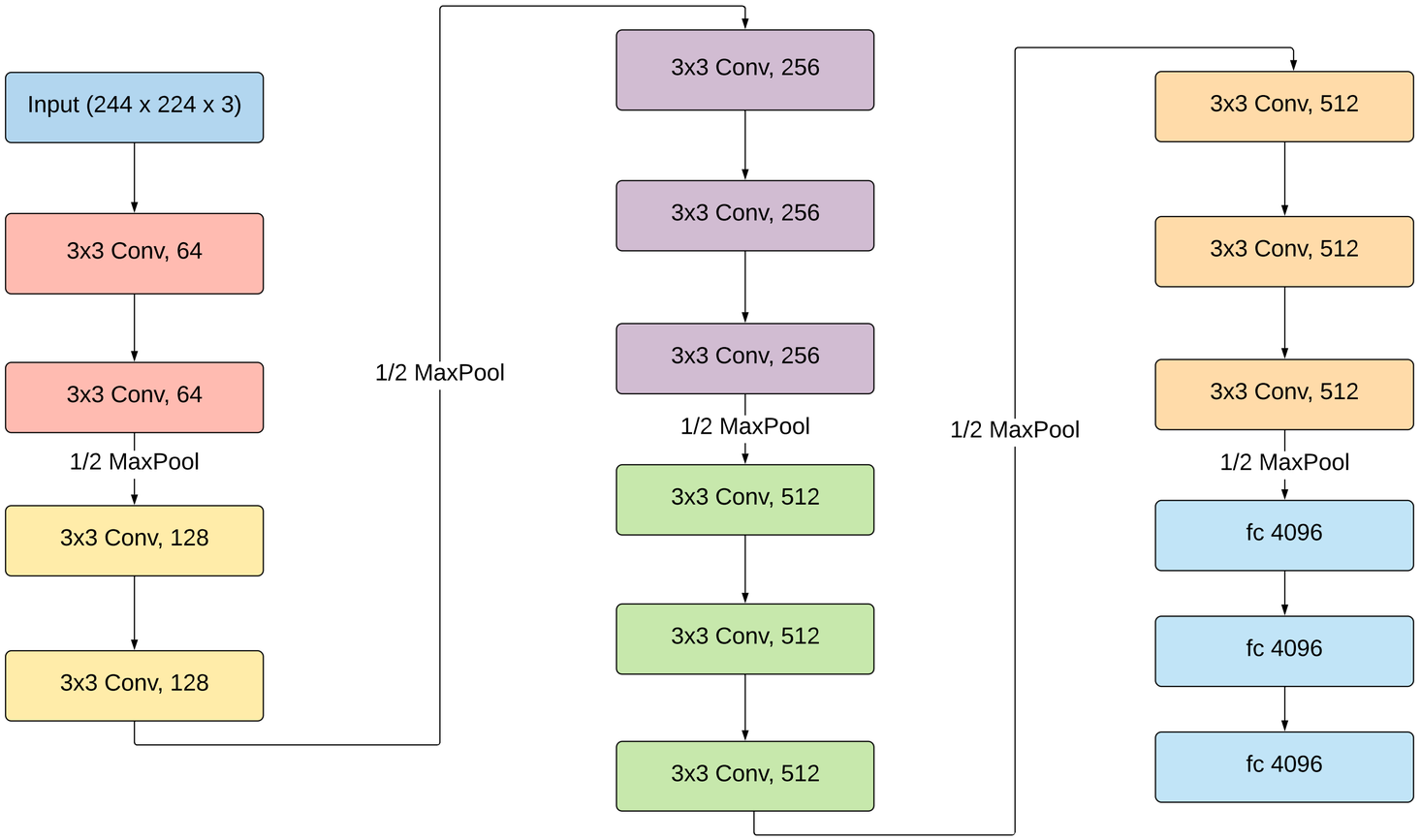
ResNet

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

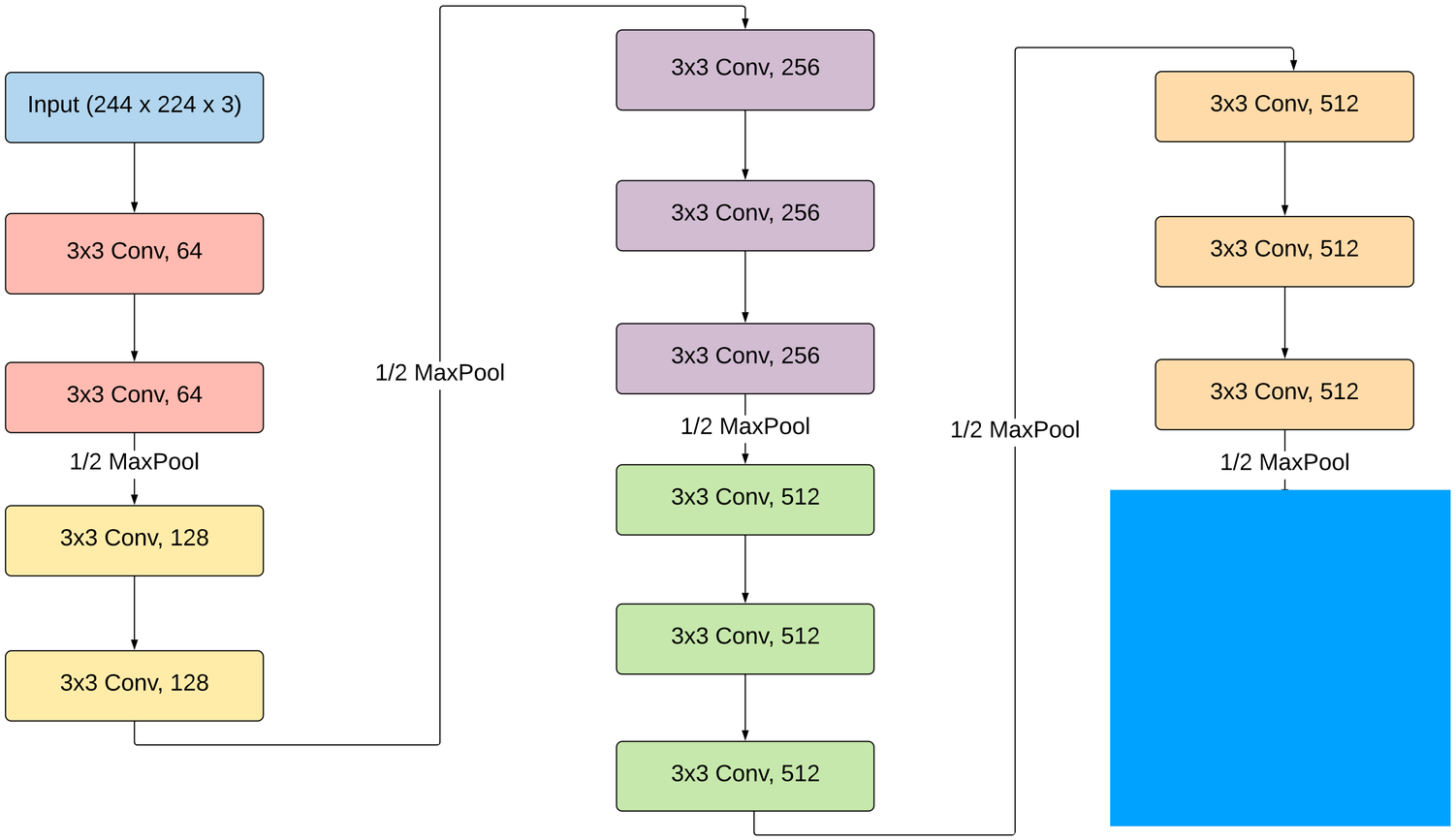




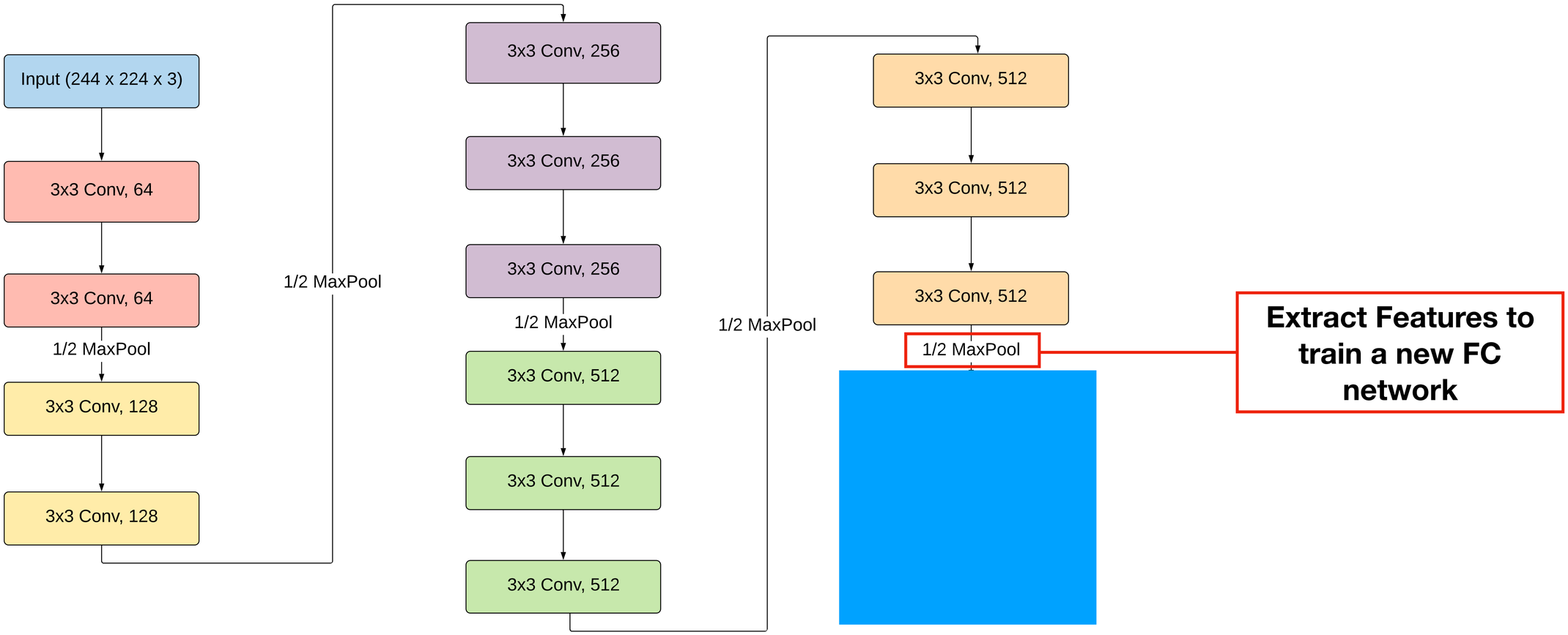
Bottleneck Features



Bottleneck Features



Bottleneck Features



Bottleneck Features

In [ ]:

```
import keras
from keras import applications
from keras import models
from keras.layers import Dropout, Flatten, Dense
import numpy as np

batch_size = 16

# Load the VGG16 network with the pretrained imagenet weights. Don't include the f
ully connected layers.

model = applications.VGG16(include_top=False, weights='imagenet')

# Generate feature map for training data
generator = datagen.flow_from_directory(
        'data/train',
        target_size=(150, 150),
        batch_size=batch_size,
        class_mode=None,
        shuffle=False)

bottleneck_features_train = model.predict_generator(generator, 2000//batch_size)

# save the output as a Numpy array
np.save('bottleneck_features_train.npy', bottleneck_features_train)
```

Load VGG16 model with pertained weights and no FC layers

Generate bottleneck features

Train FC layer

In [29]:

```
from keras.utils.np_utils import to_categorical
datagen_top = ImageDataGenerator(rescale=1./255)
generator_top = datagen_top.flow_from_directory(
         train_data_dir,
         target_size=(img_width, img_height),
         batch_size=batch_size,
         class_mode='categorical',
         shuffle=False)

num_classes = len(generator_top.class_indices)

# load the bottleneck features saved earlier
train_data = np.load('bottleneck_features_train.npy')

# get the class lebels for the training data, in the original order
train_labels = generator_top.classes
```

Load bottleneck features for training

Load labels

Train FC layer

In [ ]:

```
model = Sequential()
model.add(Flatten(input_shape=train_data.shape[1:]))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

history = model.fit(train_data, train_labels,
          epochs=50,
          batch_size=batch_size,
          validation_data=(validation_data, validation_labels))
model.save_weights('bottleneck_fc_model.h5')
```

Train FC layer

In [ ]:

```
model = Sequential()
model.add(Flatten(input_shape=train_data.shape[1:]))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

history = model.fit(train_data, train_labels,
          epochs=50,
          batch_size=batch_size,
          validation_data=(validation_data, validation_labels))
model.save_weights('bottleneck_fc_model.h5')
```

Pass bottleneck features and labels to fitting

Train FC layer

In [ ]:

```
model = Sequential()
model.add(Flatten(input_shape=train_data.shape[1:]))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

history = model.fit(train_data, train_labels,
          epochs=50,
          batch_size=batch_size,
          validation_data=(validation_data, validation_labels))
model.save_weights('bottleneck_fc_model.h5')
```

Pass bottleneck features and labels to fitting

Save FC weights

Fine Tune Network

In [65]:

```
# build model
base_model = applications.VGG16(input_shape=input_shape, include_top=False, weight
s='imagenet')

# Create a model
fullyconnected_model = Sequential()
fullyconnected_model.add(Flatten(input_shape=base_model.output_shape[1:]))
fullyconnected_model.add(Dense(256, activation='relu'))
fullyconnected_model.add(Dropout(0.5))
fullyconnected_model.add(Dense(1, activation='sigmoid'))

fullyconnected_model.load_weights('bottleneck_fc_model.h5')

model = models.Model(inputs= base_model.input, outputs= fullyconnected_model(base_
model.output))

for layer in model.layers[:-3]:
    layer.trainable = False

adam=keras.optimizers.Adam(lr=0.0001)
model.compile(optimizer=adam,
              loss='binary_crossentropy',
              metrics=['accuracy'])

print('model compiled')

model.summary()
```

Fine Tune Network

In [65]:

```
# build model
base_model = applications.VGG16(input_shape=input_shape, include_top=False, weight
s='imagenet')

# Create a model
fullyconnected_model = Sequential()
fullyconnected_model.add(Flatten(input_shape=base_model.output_shape[1:]))
fullyconnected_model.add(Dense(256, activation='relu'))
fullyconnected_model.add(Dropout(0.5))
fullyconnected_model.add(Dense(1, activation='sigmoid'))

fullyconnected_model.load_weights('bottleneck_fc_model.h5')

model = models.Model(inputs= base_model.input, outputs= fullyconnected_model(base_
model.output))

for layer in model.layers[:-3]:
    layer.trainable = False

adam=keras.optimizers.Adam(lr=0.0001)
model.compile(optimizer=adam,
              loss='binary_crossentropy',
              metrics=['accuracy'])

print('model compiled')

model.summary()
```

Load FC weights

Fine Tune Network

In [65]:

```
# build model
base_model = applications.VGG16(input_shape=input_shape, include_top=False, weight
s='imagenet')

# Create a model
fullyconnected_model = Sequential()
fullyconnected_model.add(Flatten(input_shape=base_model.output_shape[1:]))
fullyconnected_model.add(Dense(256, activation='relu'))
fullyconnected_model.add(Dropout(0.5))
fullyconnected_model.add(Dense(1, activation='sigmoid'))

fullyconnected_model.load_weights('bottleneck_fc_model.h5')

model = models.Model(inputs= base_model.input, outputs= fullyconnected_model(base_
model.output))

for layer in model.layers[:-3]:
    layer.trainable = False

adam=keras.optimizers.Adam(lr=0.0001)
model.compile(optimizer=adam,
              loss='binary_crossentropy',
              metrics=['accuracy'])

print('model compiled')

model.summary()
```

Load FC weights

Combine VGG16 base with the FC network

Fine Tune Network

In [65]:

```
# build model
base_model = applications.VGG16(input_shape=input_shape, include_top=False, weight
s='imagenet')

# Create a model
fullyconnected_model = Sequential()
fullyconnected_model.add(Flatten(input_shape=base_model.output_shape[1:]))
fullyconnected_model.add(Dense(256, activation='relu'))
fullyconnected_model.add(Dropout(0.5))
fullyconnected_model.add(Dense(1, activation='sigmoid'))

fullyconnected_model.load_weights('bottleneck_fc_model.h5')

model = models.Model(inputs= base_model.input, outputs= fullyconnected_model(base_
model.output))

for layer in model.layers[:-3]:
    layer.trainable = False

adam=keras.optimizers.Adam(lr=0.0001)
model.compile(optimizer=adam,
              loss='binary_crossentropy',
              metrics=['accuracy'])

print('model compiled')

model.summary()
```

Load FC weights

Combine VGG16 base with the FC network

Chose which layers to freeze

Layer (type)	Output Shape	Param #
input_11 (InputLayer)	(None, 150, 150, 3)	0
block1_conv1 (Conv2D)	(None, 150, 150, 64)	1792
block1_conv2 (Conv2D)	(None, 150, 150, 64)	36928
block1_pool (MaxPooling2D)	(None, 75, 75, 64)	0
block2_conv1 (Conv2D)	(None, 75, 75, 128)	73856
block2_conv2 (Conv2D)	(None, 75, 75, 128)	147584
block2_pool (MaxPooling2D)	(None, 37, 37, 128)	0
block3_conv1 (Conv2D)	(None, 37, 37, 256)	295168
block3_conv2 (Conv2D)	(None, 37, 37, 256)	590080
block3_conv3 (Conv2D)	(None, 37, 37, 256)	590080
block3_pool (MaxPooling2D)	(None, 18, 18, 256)	0
block4_conv1 (Conv2D)	(None, 18, 18, 512)	1180160
block4_conv2 (Conv2D)	(None, 18, 18, 512)	2359808
block4_conv3 (Conv2D)	(None, 18, 18, 512)	2359808
block4_pool (MaxPooling2D)	(None, 9, 9, 512)	0
block5_conv1 (Conv2D)	(None, 9, 9, 512)	2359808
block5_conv2 (Conv2D)	(None, 9, 9, 512)	2359808
block5_conv3 (Conv2D)	(None, 9, 9, 512)	2359808
block5_pool (MaxPooling2D)	(None, 4, 4, 512)	0
sequential_13 (Sequential)	(None, 1)	2097665

Total params: 16,812,353
Trainable params: 4,457,473
Non-trainable params: 12,354,880

From Scratch

	precision	recall	f1-score	support
dog	0.84	0.63	0.72	400
cat	0.71	0.88	0.78	400
avg / total	0.77	0.76	0.75	800

Using VGG16 Bottelneck Features

	precision	recall	f1-score	support
dog	0.88	0.87	0.87	400
cat	0.87	0.88	0.88	400
avg / total	0.88	0.88	0.87	800

Fine Tuning VGG16

```
Found 800 images belonging to 2 classes.
```

	precision	recall	f1-score	support
dog	0.93	0.93	0.93	400
cat	0.93	0.94	0.93	400
avg / total	0.93	0.93	0.93	800