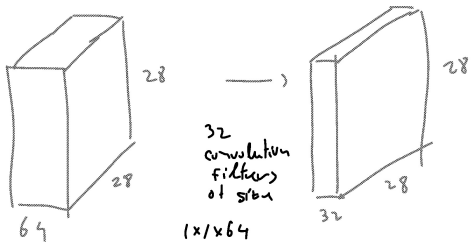
Convolution layers

- 1x1 convolution is used to reduce dimensions (i.e. lower the number of channels)

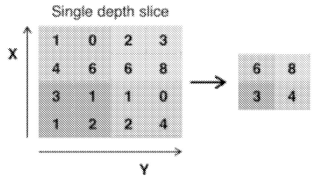


In middle of the convolution layer models, if we wanted to reduce the complexity of the n/w and would like to reduce the dimension, we can just apply 1x1 filter which would just apply the convolution across the channels based on the number of filters used keeping the spatial dimensions same (28x28)

combines channels instead of the spatial neighborhood

Pooling

- Pooling = down sampling spatial dimensions (depth unchanged)
- Max pooling: partition non overlapping regions and choose max in each region
- Using 2 x 2 regions reduces the layer dimensions by 75%
- Alternatives:
  - Average pooling
  - L2 norm pooling
  - ROI pooling (output size is fixed and input size is variable)



We wanted to have an overview of the image while we go through layers , so we need a pyramid to have greater receptive area.

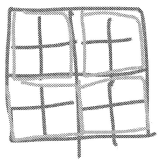
this is achieved by pooling/stride convolution.

In pooling, we down sample the image...it is thought to be of the filter and the stride is the filter size so that the filter placements dont overlap thereby reducing the spatial dimensions by half.

In conv, we take the weighted sum but here in pooling we may use the average, max,etc.. Average is not much preferred because the filter is not learnt and there s no learning here in just taking average of the neighborhood.....so the popular one is max pooling

Pooling

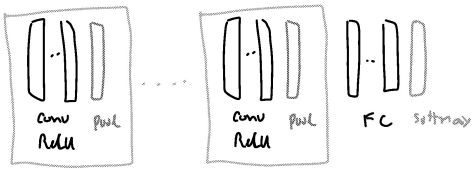
- Pooling is done by scanning with a filter with stride
- Stride is normally selected without filter overlap (downsampling). For example a 2 x 2 filter with a stride of 2
- Convolution with a stride can also be used to downsample but this will average and so we normally downsample with pooling instead of convolution
- Pooling retains more information (e.g. indicates if a feature is there or not)



Pooling

- Pulling is a layer without parameters and has no learning
- The depth dimension is normally not pooled
- Pulling supports multiple scale analysis (a 3x3 window in a pooled layer covers a larger area in the layer before it)
- Pulling helps in reducing the number of network coefficients
- The amount of pooling is a design choice (hyperparameters)

Example network:



Keras MNIST example

```
from keras import layers
from keras import models
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.summary()
```

```
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
model.summary()
```

softmax for getting the probability of the 10 classes

Convolution parameters

```
Conv2D(output_depth, (window_height, window_width))
```

strides = 1, padding = 'valid' ('valid': no padding, 'same': yes padding)
dilation_rate = 1

dilated conv

Keras MNIST example

No.of parameters needed to arrive this layer

input ->28x28x1 (greyscale)

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
maxpooling2d_1 (MaxPooling2D) (*)  (None, 13, 13, 32)  0
_________________________________________________________________
conv2d_2 (Conv2D)            (**) (None, 11, 11, 64)   18496
_________________________________________________________________
maxpooling2d_2 (MaxPooling2D) (*)  (None, 5, 5, 64)    0
_________________________________________________________________
conv2d_3 (Conv2D)            (**) (None, 3, 3, 64)     36928
_________________________________________________________________
flatten_1 (Flatten)          (None, 576)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                36928
_________________________________________________________________
dense_2 (Dense)              (None, 10)                650
=================================================================
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
```

(*) Downsample
(**) Reduction by 2 due to no zero padding

32 3x3x1 filters (32 x (3x3x1+1))
64 3x3x32 filters (64 x (3x3x32+1))
64 3x3x64 filters (64 x (3x3x64+1))
3x3x64
64 units with 576 inputs (64 x (576+1))
10 units with 64 inputs (10 x (64+1))

Nonex26x26x32---->
None - actually would represent the batch size
26x26 -->reduced by 2 from 28x28 bcz no padding
32 -- no.of filters

Conv -->reduce the information from spatial dimention to bringing the pyramid view along with the increase in the number of channels thereby collecting all the feature details/data

Keras MNIST example

* Train on MNIST:

```
from keras.datasets import mnist
from keras.utils import to_categorical

# load data
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype('float32') / 255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

# compile model, fit, and evaluate
model.compile(optimizer='rmsprop', loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=5, batch_size=64)
test_loss, test_acc = model.evaluate(test_images, test_labels)
```

```
test_acc                # 0.9908000000000001
```

Convnets improved the 97.8% accuracy of a fully connected network to 99.1%

MNIST-->handwritten digits

Why is MNIST problem simple and easy--->because the images are cropped to 1 digit and there is no localization needed much

Keras cats/dogs example

- **Cats and dogs classification (Kaggle challenge)**
- **We use a subset of 2000 cats + 2000 dogs**
- **Larger image size (150 x 150) compared with MNIST (28 x 28) and hence there's a need for a deeper network**

```
from keras import layers
from keras import models
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
model.summary()
```

same number of filters/channels used before and after, the middle pooling layer is only to control the complexity by bringing down the spatial dim

binary classification

Keras cats/dogs example

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 148, 148, 32)      896
_________________________________________________________________
maxpooling2d_1 (MaxPooling2D (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 72, 72, 64)        18496
_________________________________________________________________
maxpooling2d_2 (MaxPooling2D (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 34, 34, 128)       73856
_________________________________________________________________
maxpooling2d_3 (MaxPooling2D (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 15, 15, 128)       147584
_________________________________________________________________
maxpooling2d_4 (MaxPooling2D (None, 7, 7, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 6272)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               3211776
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 513
=================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
```

bigger images making the number of parameters high in fully connected layer

Keras cats/dogs example

* Compile network:

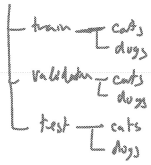
```
from keras import optimizers
model.compile(
    loss='binary_crossentropy',
    optimizer=optimizers.RMSprop(lr=1e-4),
    metrics=['acc'])
```

* Pre-process data:
  - Load and rescale to [0..1] using a keras data generation

Keras cats/dogs example

* pre-process data using ImageDataGenerator:

```
from keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(
    train_dir,                  # image folder
    target_size=(150, 150),     # resize images
    batch_size=20,
    class_mode='binary')           .
validation_generator = test_datagen.flow_from_directory(
    validation_dir,
    target_size=(150, 150),
    batch_size=20,
    class_mode='binary')
```



- Generate 20 x 150 x 150 x 3 tensors for training and validation and corresponding binary labels

Keras cats/dogs example

* Fit model:
  Use `fit_generator` instead of `fit`
  - The # of batches of size `batch_size` to complete a complete epoch is:

$$\text{steps\_per\_epoch} = \frac{\#\text{ examples}}{\text{batch\_size}} = \frac{2000}{20} = 100$$

```
history = model.fit_generator(
    train_generator,
    steps_per_epoch=100,
    epochs=30,
    validation_data=validation_generator,
    validation_steps=50)
model.save('cats_and_dogs_small_1.h5')
```

instead of loading the data from keras, if we do in realtime from our files-->we will have to used data generator whic helps to prepare the batches needed for training and it also scales the images of different resolution to the same size

Epoch: An epoch is one complete iteration over the entire training dataset. After an epoch, the model has seen all training samples once.

Batch Size: This is the number of samples processed before the model's internal parameters are updated. Smaller batch sizes lead to more updates per epoch.

steps per epoch --> how many times there would be an update to the parameters in one epoch. So, if there are 1000 samples in training set and we would like to update the paramaeters after seeing 100 samples(batch size) every time, then the no.of times the parameters are updated is given by division

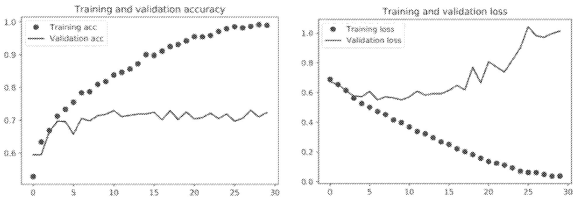
number of epochs is generally not fixed by user-- we say that if the validation loss stopped decreasing then we would stop training because there is no point in training --there is no further learning

Keras cats/dogs example

* plot results.

```
import matplotlib.pyplot as plt
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)

# Plot accuracy
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

# Plot loss
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```

Keras cats/dogs example



- overfit after 5 iterations due to small dataset
67% validation accuracy

validation loss increases very quickly because we overfit -->why wuickly?--smaller dataset--easily overfitted

Preventing overfiiting methods:
1. early stopping - stop training early itself
2. data augmentation-->take images in the smaller dataset and tansform it by rotation,scaling,etc ..to show like it has a lot of images/data
3. Increase Regularization
4. Transfer learning -- take pretrained weights from a diff problem
5. Add dropout

Keras cats/dogs example

* To further prevent overfitting, add dropout:

```
# define a new convnet that includes dropout
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dropout(0.5))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['acc'])
```

Dropout - > during the training, some of the neuron inputs stops doing work/contribution-->they are dropped off from learning path-->thereby generalising the logic

Keras cats/dogs example

- **To address the problem with limited data:**
  1. **Data augmentation: add examples with perturbations (e.g., rotation, flip, contrast change)**
  2. **Transfer learning: use a pre-trained convolution base**
- **Freeze loaded weights of pre-trained blocks after loading them**

Refer image

Data augmentation

Data Augmentation

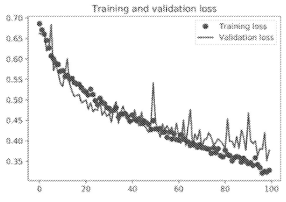
```
train_datagen = ImageDataGenerator(                    # different from before
rescale=1./255,
rotation_range=46,
width_shift_range=0.2,
height_shift_range=0.2,
shear_range=0.2,
zoom_range=0.2,
horizontal_flip=True,)
test_datagen = ImageDataGenerator(rescale=1./255)      # as before

train_generator = train_datagen.flow_from_directory(   # as before
    train_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary')
validation_generator = test_datagen.flow_from_directory( # as before
    validation_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary')

history = model.fit_generator(                         # as before
    train_generator,
    steps_per_epoch=100,
    epochs=100,
    validation_data=validation_generator,
    validation_steps=50)

model.save('cats_and_dogs_small_2.h5')                 # as before
```

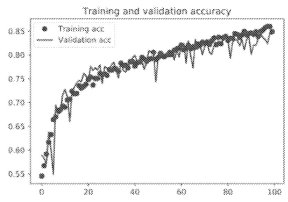
no augmentation during testing on validation

offline vs online -->
offline-->augment the data and save it offline and use it in the training data
online-->augment on the go..random augmentaiton for every batch and we get infinite number if combinations of the data-->more generic results.
helps in reducing the storage offline
disadv--more computation

we only wan to test on original/true data....if suppose the augment method we made was wrong , it could be easily found on testing proper samples than the augmented test sample itself

Data augmentation

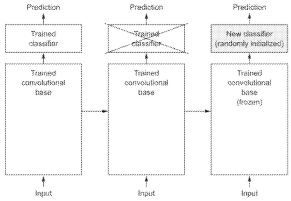




- does not overfit. 82% validation accuracy
- Additional improvement is possible with hyperparameter tuning.

Transfer learning

- **Use a pre-trained convnet trained on a large data set (e.g., ImageNet object classification)** Imagenet has a lot data-->1.4M images
- **ImageNet: 1.4 million labeled images, 1000 classes (animals, objects)**
- **Use the convolution layers of VGG16 to extract features for the cat/dog problem (representation learning)**
- **Whether to use higher convolution layers depends on how similar the data sets are**



imagenet has lot of data ...it may have cats/dogs as well but there would be a lot more irrelevant images for our problem, but still we use for the purpose of the layers thar know how to extract s feature

initialize almost zero --->since this is almost zero, if we send the gradients to the below trained n/w, we would destroy the trained weights, so we freeze the below layers and dont train them.just train the upper layers specific to this problem alone.

When the upper layers are trained , we then unfreeze the lower part and then train all of them together entriely, to finetune the below layers as well to this problem

Transfer learning

* Pre-trained models available in Keras:

Models for image classification with weights trained on ImageNet:

- Xception
- VGG16
- VGG19
- ResNet, ResNetV2, ResNeXt
- InceptionV3
- InceptionResNetV2
- MobileNet
- MobileNetV2
- DenseNet
- NASNet

Transfer learning

* Load VGG16

```
from keras.applications import VGG16
conv_base = VGG16(
    weights='imagenet',         # weights checkpoint from which to initialize model
    include_top=False,          # do not include the fully connected layers
                                # (responsible for classifying 1000 classes)
    input_shape=(150, 150, 3))  # optional
conv_base.summary()
```

Refer summary in images

this whole bunch of conv/max pool layer will be one big layer in our problem model.
the input imageneed not match the input on which this vgg was trined bcz we actually define only the conv filters in the lsyers and it has nothing specific to its original image, it would work well on the other input sizes as well except that the ouputs after each layer would vary bassed on the current input

no issues with input as long as we dont use the flatten layer from the trained one-->becasue the size would vary and so does the parameters

Transfer learning

- **Add pre-trained layers to the network:**
  - **Add conv-base (the loaded model) as a layer**
  - **Freeze weights of pre-trained network**
  - **Train end-to-end**
- **Larger and slower network**

```
from keras import models
from keras import layers
model = models.Sequential()
model.add(conv_base)
model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
model.summary()
```

Transfer learning

```
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 4, 4, 512)         14714688
_________________________________________________________________
flatten_1 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               2097408
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 257
=================================================================
Total params: 16,812,353
Trainable params: 16,812,353
Non-trainable params: 0
```

we dont train them intiallly-->freeze it -->see below

Transfer learning

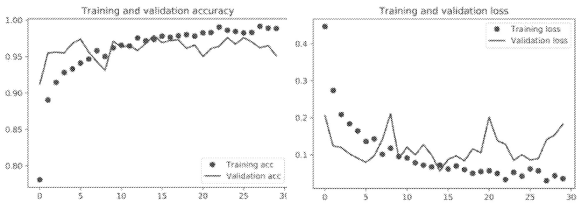
- **Freeze the loaded model so as to not to destroy the weights by gradients from untrained fully connected layers on top**

```
conv_base.trainable = False
```

```
print('Number of weight tensors ' 'before freezing the conv base:',
      len(model.trainable_weights))
#   Number of weight tensors before freezing the conv base: 30

conv_base.trainable = False ####
print('Number of weight tensors ' 'after freezing the conv base:',
      len(model.trainable_weights))
#   Number of weight tensors after freezing the conv base: 4
```

- **There are two weight tensors per layer: weight matrix, bias vector**

Transfer learning



- Validation accuracy of 96% with a very small data set
- Less overfitting (due to data augmentation)

Transfer learning

- Fine tuning:
  - After training the fully connected layers, unfreeze some top layers in the conv-base and retrain to allow the model to fit the data
- Steps:
  1. Add custom network on top of the trained layers
  2. Freeze the trained layers
  3. Train the custom network
  4. Unfreeze the top layers in the base network
  5. Jointly train the custom network and unfrozen layers

top layers means later layers of the trained bunch of copied layers which are closer to the layers specific to our problem's layer

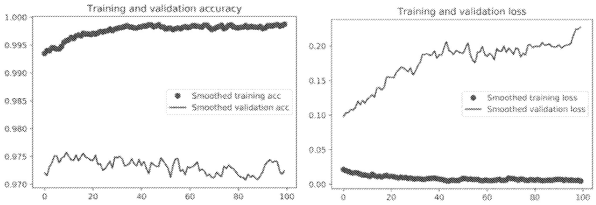
Refer images for TL summary

Transfer learning

- Freezing all layers up to block number 5:
  - Step through all the layers
  - Set “trainable“ starting with block5_conv1

```
conv_base.trainable = True
set_trainable = False
for layer in conv_base.layers:
    if layer.name == 'block5_conv1':
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False
```

Transfer learning



* Smoothed curves: 97% accuracy (1% improvement)
* Final test accuracy:

```
test_loss, test_acc = model.evaluate_generator(
    test_generator,
    steps=50)

print('test acc:', test_acc) #  97%
```

Gradual training

- Given a deep network:
  - Train of shallow network
  - Freeze the trained layers
  - Add layers
  - Retrain
  - Unfreeze and retrain all layers
- Advantage: constrain the search space and so converge better
- Disadvantage: because of limited solution space possibly larger error
- For example this procedure was used in VGG16 (but is less common now)