ML@B Bootcamp - Convolutional Neural Networks in Tensorflow

Machine Learning at Berkeley

Agenda

Why Use CNNs?

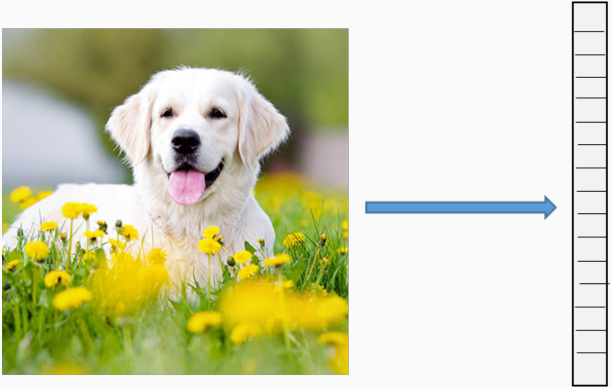
CNN Layers

Tensorflow Syntax

Transfer Learning

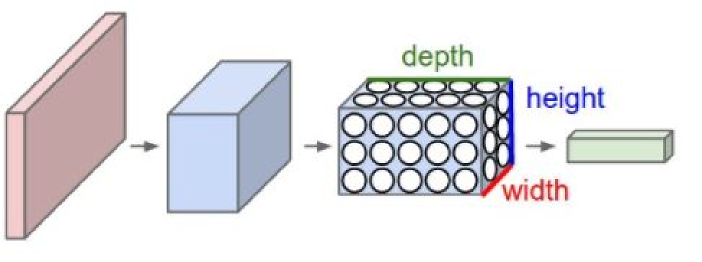
Why Use CNNs?

CNNs for Spatial Data



- Spatial information is lost if image is converted to vector

CNNs for Spatial Data



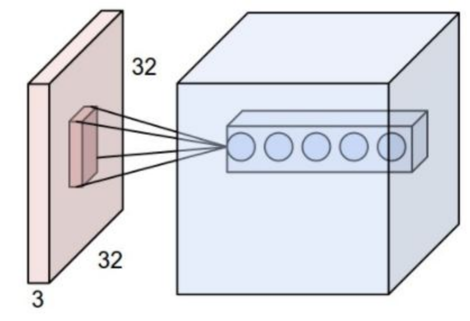
- CNNs arrange each layer of neurons into 3 dimensions

Fully Connected Networks Don't Scale for Large Images

- A 200x200 color image has 200*200*3 = 120,000 pixels
- Fully connected network will have 120,000 weights in first layer
- Having such huge number of parameters is wasteful for memory and training, and leads to overfitting

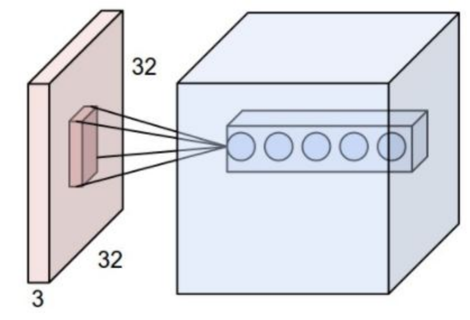
CNN Layers

Convolutional Layer



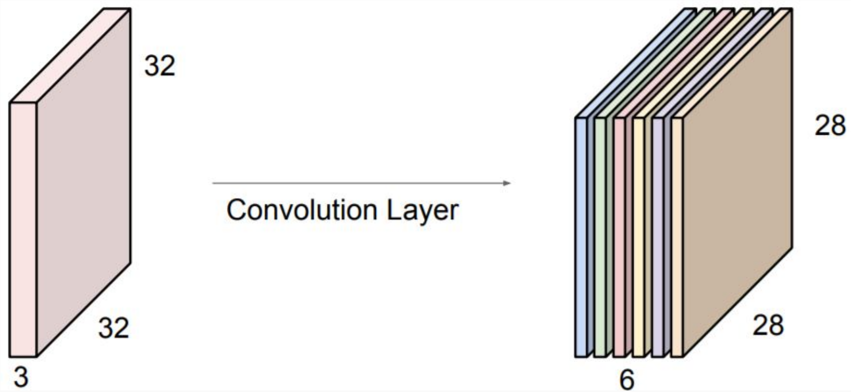
- Suppose input image has dimension 32x32x3, and each of 5 filters has dimension 5x5x3
- Convolution operation is computing dot product of each filter along each position of the image

Convolutional Layer

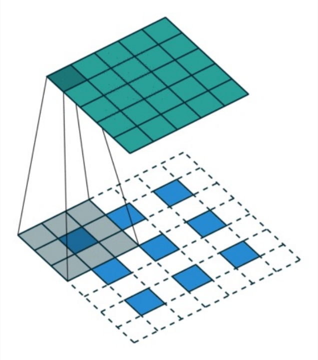


- Each filter only looks at small region of image but applies over all channels
- Multiple filters look at same region of image

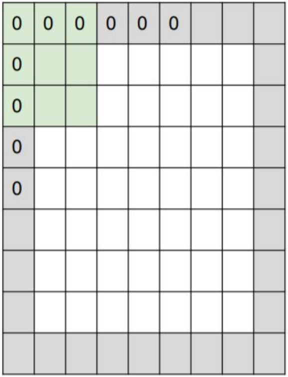
Convolutional Layer - Hyperparameters



- **Depth of Output Volume** - number of filters used on each input layer



- **Stride** - how many pixels in each dimension to move the filter



- **Zero Padding** - helps control height and width of output volume

Convolutional Layer - Calculating Output Volume

- If a square input layer has dimensions $W_1 x W_1 x D_1$ and N square filters of dimensions $FxFxD_1$ are applied with zero padding P and stride S, output volume $W_2 x W_2 x D_2$ is given by:

$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$
$$D_2 = N$$

- Eg: If input volume has size 32x32x3 and 10 5x5 filters are applied with stride 1 and zero pad 2, what is output volume?

Convolutional Layer - Calculating Output Volume

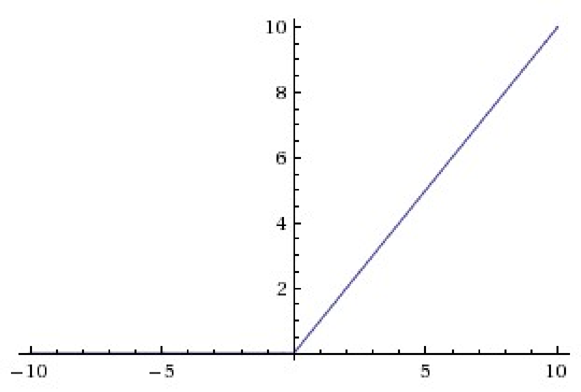
- If a square input layer has dimensions $W_1 x W_1 x D_1$ and N square filters of dimensions $FxFxD_1$ are applied with zero padding P and stride S, output volume $W_2 x W_2 x D_2$ is given by:

$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

$$D_2 = N$$

- Eg: If input volume has size 32x32x3 and 10 5x5 filters are applied with stride 1 and zero pad 2, what is output volume?
- $W_2 = \frac{32 - 5 + 2 * 2}{1} + 1 = 32$ so output volume is 32x32x10

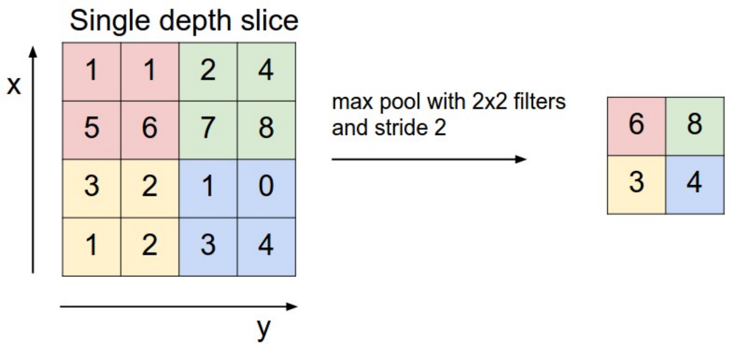
ReLU



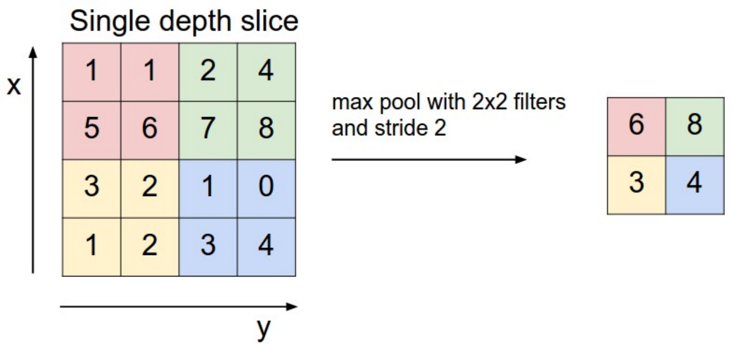
- Activation function which adds nonlinearity

Pooling Layer

- Reduces size of representation
- Applied on each channel independently instead of acting over depth of input



Pooling Layer



- Takes in input of size $W_1 x W_1 X D_1$ and applies filter of size FxF with stride S to produce output of size $W_2 x W_2 x D_2$ where $D_2 = D_1$ and $W_2 = \frac{W_1 - F}{S} + 1$

Tensorflow Syntax

tf.nn.conv2d

tf.nn.conv2d

```
conv2d(
    input,
    filter,
    strides,
    padding,
    use_cudnn_on_gpu=None,
    data_format=None,
    name=None
)
```

- input: 4d tensor of shape [number of examples in batch, height of input, width of input, number of channels of input]
- filter: 4d tensor of shape [filter height, filter width, number of input channels, number of output channels]
- Note: number of output channels = number of filters used

tf.nn.conv2d

tf.nn.conv2d

```
conv2d(
    input,
    filter,
    strides,
    padding,
    use_cudnn_on_gpu=None,
    data_format=None,
    name=None
)
```

- strides: 1D tensor of length 4 of format [1, stride vertical, stride horizontal, 1]
- padding: "SAME" or "VALID"

tf.layers.conv2d

```
tf.layers.conv2d(inputs, filters, kernel_size, strides=(1, 1), padding='valid',
data_format='channels_last', dilation_rate=(1, 1), activation=None,
use_bias=True, kernel_initializer=None,
bias_initializer=tf.zeros_initializer(), kernel_regularizer=None,
bias_regularizer=None, activity_regularizer=None, trainable=True, name=None,
reuse=None)
```

- input: 4d tensor of shape [number of examples in batch, height of input, width of input, number of channels of input]
- filters: integer of number of filters in convolution or depth of output space
- kernel_size: integer or tuple specifying width and height of filter

tf.nn.conv2d vs tf.layers.conv2d

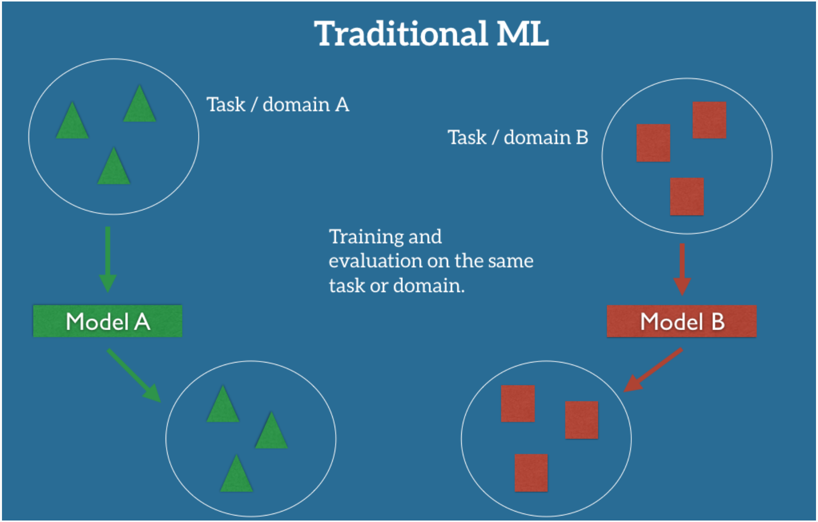
- weight tensor must be initialized in tf.nn.conv2d whereas tf.layers.conv2d only needs size of filter and output
- tf.layers.conv2d may be easier when creating and training model from scratch
- tf.nn.conv2d may be easier when loading pretrained model

Transfer Learning

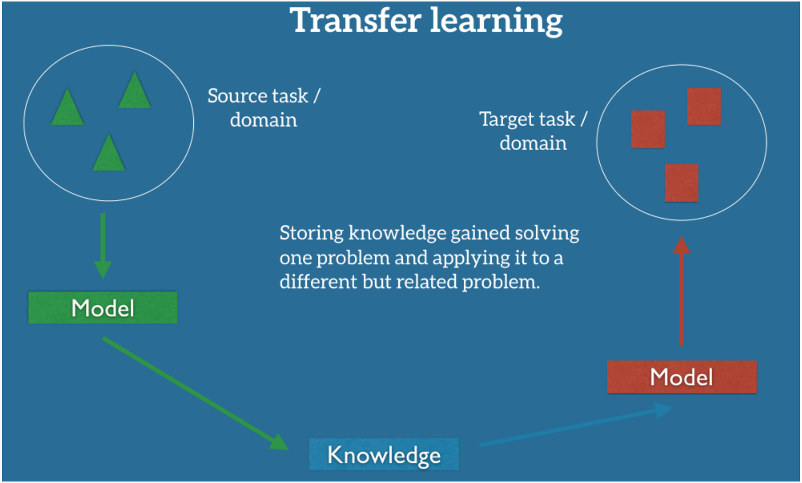
What is Transfer Learning

- Transfer learning takes the knowledge of pretrained networks and adapts them to new applications
- For example, if you want to classify domain-specific objects (say different species of birds) in images you might want to start with VGG16 (trained on imagenet)
- Imagenet isn't focused on species of birds (and thus VGG16) isn't focused on birds, but it's (much) better than nothing

Traditional ML vs. Transfer Learning



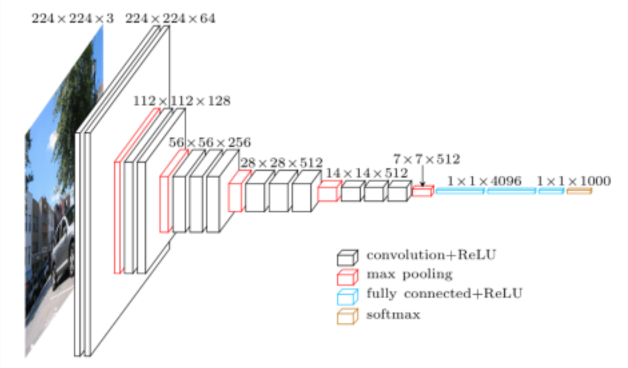
Traditional ML vs. Transfer Learning



Benefits of Transfer Learning

- Don't have to reinvent the wheel
- Is a lot faster than training a whole new model from scratch
- Don't have to gather as much data

How to Actually Transfer Learn



- Freeze the convolutional layers with imagenet weights
- Train the "top" three FC layers

Other Interpretations of Transfer Learning

- Transfer learning can also be viewed as training a "feature extractor"
- For example, VGG16 takes a 224×224 image and outputs a 512×1 feature vector

Transfer Learning Demo

- Goal: Classify cats vs dogs
- Take a pretrained convnet
- Remove the last fully-connected layer
- Extract the features
- Train a classifier

Data Augmentation

- Data Augmentation: rotation, shearing, random cropping, flipping, stretching

```
train_datagen =  ImageDataGenerator(
    preprocessing_function=preprocess_input,
    rotation_range=30,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)
```

```
train_generator = train_datagen.flow_from_directory(
  args.train_dir,
  target_size=(IM_WIDTH, IM_HEIGHT),
  batch_size=batch_size,
)
```

Transfer Learning Demo Fill in the Blanks

- Initialize Inception V3 network from keras.applications module
- Add last layer to convnet
    - Input is output of base model
    - Add pooling layer with GlobalAveragePooling2D()
    - Add fully connected Dense() layer of size 1024 with a softmax function
    - Return new model with input base model and output last layer
- Freeze all layers and compile the new model

Tips for Transfer Learn

- If new dataset is small and similar to original $\Rightarrow$ Retrain final linear classifier
- If new dataset is large and similar to original $\Rightarrow$ Fine tune through whole network
- If new dataset is small and different from original $\Rightarrow$ Retrain later layers in network
- If new dataset is large and different from original $\Rightarrow$ Use weights of previous network to initialize new network

Fine Tuning

- Replace and retrain classifier on top of convnet
- Note: Pretrained network's weights are tuned
- When to fine tune

	Similar dataset	Different dataset
Small dataset	Transfer learning: highest level features + classifier	Transfer learning: lower level features + classifier
Large dataset	Fine-tune*	Fine-tune*

Fine Tuning Demo Fill in the Blanks

- Freeze all layers and compile model
    - Set each layer.trainable = False
    - Compile model with rmsprop optimizer, categorical cross entropy loss, and accuracy metrics
    - Lower learning rate from learning rate used during training from scratch

Answers 1

- Initialize Inception V3 network from keras.applications module

```
base_model = InceptionV3(weights='imagenet', include_top=False)
```

- Add last layer to convnet

```
def add_new_last_layer(base_model, nb_classes):
  """Add last layer to the convnet
  Args:
    base_model: keras model excluding top
    nb_classes: # of classes
  Returns:
    new keras model with last layer
  """
  x = base_model.output
  x = GlobalAveragePooling2D()(x)
  x = Dense(FC_SIZE, activation='relu')(x)
  predictions = Dense(nb_classes, activation='softmax')(x)
  model = Model(input=base_model.input, output=predictions)
  return model
```

- Freeze all layers and compile the new model

```
def setup_to_transfer_learn(model, base_model):
  """Freeze all layers and compile the model"""
  for layer in base_model.layers:
    layer.trainable = False
  model.compile(optimizer='rmsprop',
```

Answers 2

- Freeze all layers and compile the new model

```
def setup_to_transfer_learn(model, base_model):
  """Freeze all layers and compile the model"""
  for layer in base_model.layers:
    layer.trainable = False
  model.compile(optimizer='rmsprop',
                loss='categorical_crossentropy',
                metrics=['accuracy'])
```

- Finetuning: Freeze all layers and compile model

```
def setup_to_finetune(model):
   """Freeze the bottom NB_IV3_LAYERS and retrain the remaining top
      layers.
   note: NB_IV3_LAYERS corresponds to the top 2 inception blocks in
         the inceptionv3 architecture
   Args:
     model: keras model
   """
   for layer in model.layers[:NB_IV3_LAYERS_TO_FREEZE]:
      layer.trainable = False
   for layer in model.layers[NB_IV3_LAYERS_TO_FREEZE:]:
      layer.trainable = True
   model.compile(optimizer=SGD(lr=0.0001, momentum=0.9),
                 loss='categorical_crossentropy')
```

Prediction

```
python predict.py --image dog.001.jpg --model dc.model
python predict.py --image_url https://goo.gl/Xws7Tp --model dc.model
```