https://towardsdatascience.com/the-quest-of-higher-accuracy-for-cnn-models-42df5d731faf

Week 4-5 Research

Maria Shevchuk

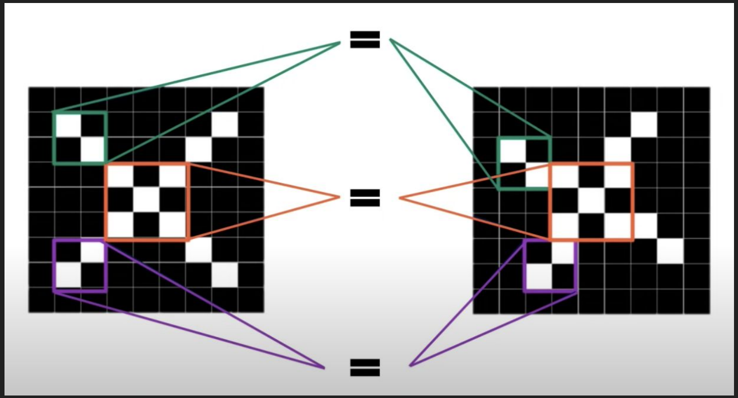
What does the number of filters in a convolution layer convey?

of filters in a convolution layer

- Filter = set of weights applied to the input image

  = feature detector

- So number of filters = number of features to detect
- More filters = more learning
  - Not always a good thing…



How are filters defined in a CNN?

They are not :)

What about the size of the filter/kernel? Is there a way to choose the best size?

NO :)

Kernel Size

- Larger (5x5 ...) kernel sizes
  - consume lots of time in training
- Smaller kernels (1x1, 2x2, 3x3, 4x4)
  - Reduces computational costs and weight sharing
    - Lesser weights for backpropagation
- 3x3 kernel size
  - Introduced in 2015 (VGG CNNs)
- Why 3x3?
  - 1x1: just one pixel, no info from neighboring pixels
  - 2x2 / 4x4: odd-sized filters are preferred (symmetrical divide)
    - If no symmetry → distortions

Simple CNN Implementation

```python
import numpy as np
import mnist

train_images = mnist.train_images()
train_labels = mnist.train_labels()
test_images = mnist.test_images()
test_labels = mnist.test_labels()

# Normalize the images.
train_images = (train_images / 255) - 0.5
test_images = (test_images / 255) - 0.5

# Reshape the images.
train_images = np.expand_dims(train_images, axis=3)
test_images = np.expand_dims(test_images, axis=3)

print(train_images.shape) # (60000, 28, 28, 1)
print(test_images.shape)  # (10000, 28, 28, 1)
```

*Keras requires a third dimension

Building the model

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatte

num_filters = 8
filter_size = 3
pool_size = 2

model = Sequential([
  Conv2D(num_filters, filter_size, input_shape=(28, 28, 1)),
  MaxPooling2D(pool_size=pool_size),
  Flatten(),
  Dense(10, activation='softmax'),
])
```

*First layer must always specify the input shape

*The Sequential class represents a linear stack of layers

*10 nodes, one for each class

Compile!

*Adam is a default gradient based optimizer

```
model.compile(
  'adam',
  loss='categorical_crossentropy',
  metrics=['accuracy'],
)
```

*Using categorical_crossentropy because we have >2 layers

$$L_{\mathrm{CE}} = -\sum_{i=1}^{n} t_i \log(p_i), \quad \text{for n classes,}$$

where t_i is the truth label and p_i is the Softmax probability for the i^{th} class.

Let’s train the model!

*Utility method to_categorical turns the array of class integers into an array of one-hot vectors

```
from tensorflow.keras.utils import to_categorical

model.fit(
  train_images,
  to_categorical(train_labels),
  epochs=3,
  validation_data=(test_images, to_categorical(test_labels)),
)
```

Now we test!

```
# Predict on the first 5 test images.
predictions = model.predict(test_images[:5])

# Print our model's predictions.
print(np.argmax(predictions, axis=1)) # [7, 2, 1, 0, 4]

# Check our predictions against the ground truths.
print(test_labels[:5]) # [7, 2, 1, 0, 4]
```

Keras vs PyTorch

Keras

- ➢ Good for small datasets
- ➢ Fast prototyping - very simple & readable architecture
- ➢ Most popular (though not by much)

PyTorch

- ➢ Faster!
- ➢ Large datasets, high performance
- ➢ More complex code/architecture

Next week’s Action Items

- Dropout layers and overfitting - how and why?
- Normalizing data + Lance’s Medium article
- Kviz!
- What affects network accuracy and efficiency?
    - Run some tests