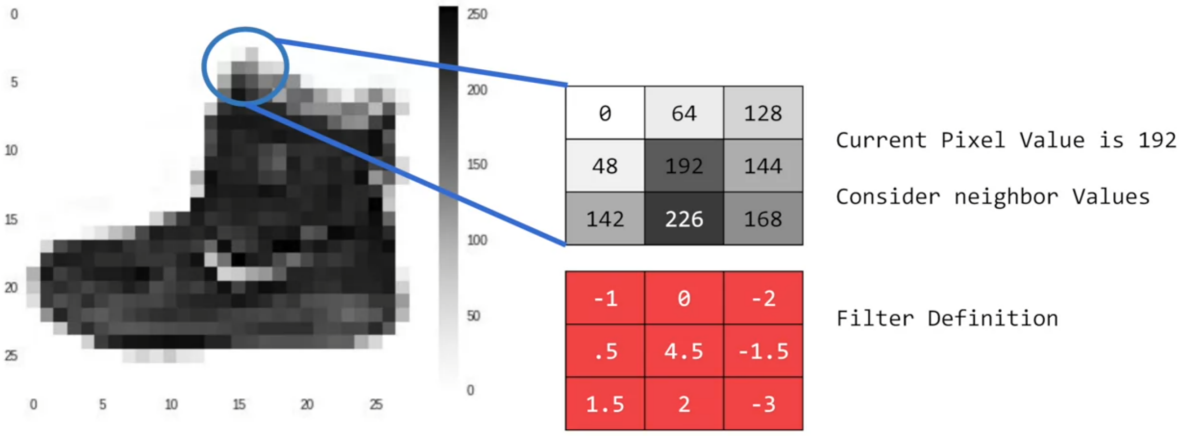
Week 3

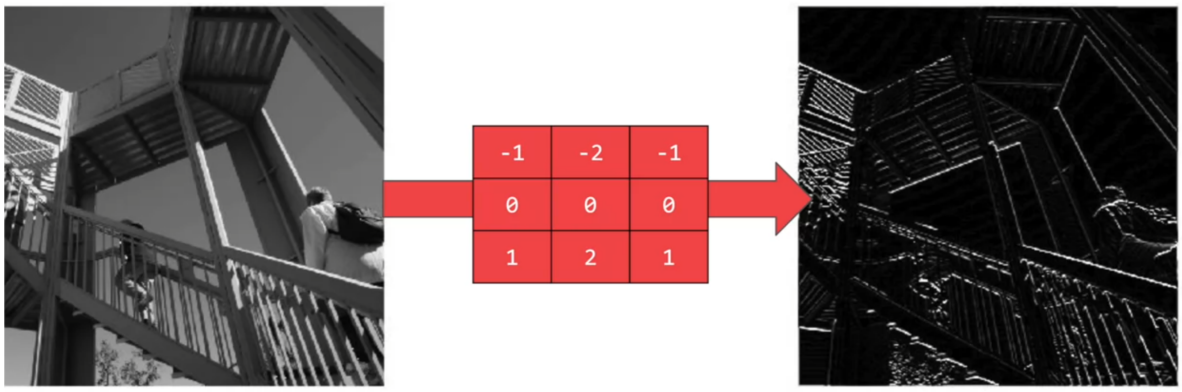
Convolutions and polling

- A lot of wasted space
- Condensing the image down to only the important features
- And the condensing is called **convolution**
  - Passing the filter into the image in order to change the underlying image
  - For every pixel and take a look at it's neighbors
  - Multiply each neighbor by the corresponding value of the filter

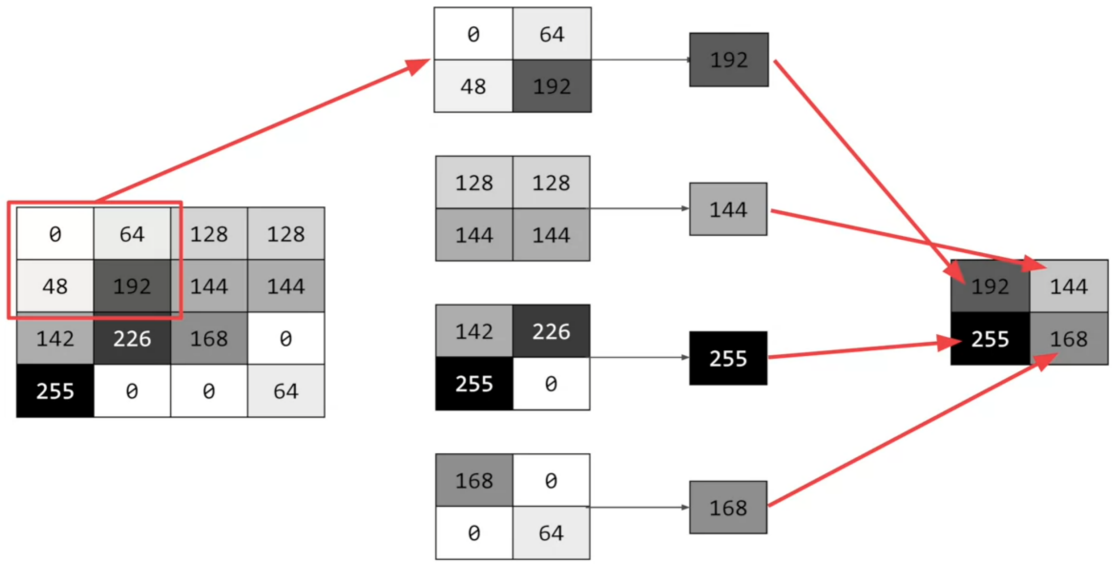



```
CURRENT_PIXEL_VALUE = 192
NEW_PIXEL_VALUE = (-1 * 0) + (0 * 64) + (-2 * 128) +
                  (.5 * 48) + (4.5 * 192) + (-1.5 * 144) +
                  (1.5 * 42) + (2 * 226) + (-3 * 168)
```

- The point of this is so that some part of the convolution will get emphasized



- Polling is compressing the image by using only the biggest value in a sample
- Example: (Using the biggest value in the 4 grayscale)



- Those are:
  - Conv2D layers
  - MaxPooling2D layers

Implementing Convolution Layers

```
model = keras.Sequential([
    keras.layers.Conv2D(64, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    keras.layers.MaxPool2D(2, 2),
    keras.layers.Conv2D(64, (3, 3), activation='relu'),
    keras.layers.MaxPool2D(2, 2),
    keras.layers.Flatten(),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])
```

- The content have been greatly simplified
- The goal is so the convolution will filter the features that determine the output
- Useful method on the model:
  - `model.summary()`
  - This allows to inspect the model and see the journey of the convolution
- Example:

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 26, 26, 64)        640
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 13, 13, 64)        0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 11, 11, 64)        36928
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 5, 5, 64)          0
_________________________________________________________________
flatten_5 (Flatten)          (None, 1600)              0
_________________________________________________________________
dense_10 (Dense)             (None, 128)               204928
_________________________________________________________________
dense_11 (Dense)             (None, 10)                1290
=================================================================
```

Improve Fashion Classifier

```
import tensorflow as tf

mnist = tf.keras.datasets.fashion_mnist
(training_images, training_labels), (test_images, test_labels) = mnist.load_data()

training_images=training_images.reshape(60000, 28, 28, 1)
training_images=training_images / 255.0

test_images = test_images.reshape(10000, 28, 28, 1)
test_images=test_images/255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(64, (3,3), activation='relu', input_shape=(28, 28, 1)),
  tf.keras.layers.MaxPooling2D(2, 2),
  tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
  tf.keras.layers.MaxPooling2D(2,2),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(10, activation='softmax')
])
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.summary()
model.fit(training_images, training_labels, epochs=5)
test_loss = model.evaluate(test_images, test_labels)
```

Visualizing Convolutions

```
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import models

class myCallback(tf.keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs={}):
    if(logs.get('accuracy')>0.95):
      print("\nReached 90% accuracy so cancelling training!")
      self.model.stop_training = True
```

```
callback = myCallback()

mnist = tf.keras.datasets.fashion_mnist
(training_images, training_labels), (test_images, test_labels) = mnist.load_data()

training_images=training_images.reshape(60000, 28, 28, 1)
training_images=training_images / 255.0

test_images = test_images.reshape(10000, 28, 28, 1)
test_images=test_images/255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(64, (3,3), activation='relu', input_shape=(28, 28, 1)),
  tf.keras.layers.MaxPooling2D(2, 2),
  tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
  tf.keras.layers.MaxPooling2D(2,2),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(10, activation='softmax')
])
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.fit(training_images, training_labels, epochs=10, callbacks=[callback])
test_loss = model.evaluate(test_images, test_labels)

f, axarr = plt.subplots(3,4)
FIRST_IMAGE=3
SECOND_IMAGE=4
THIRD_IMAGE=5
CONVOLUTION_NUMBER = 1

layer_outputs = [layer.output for layer in model.layers]
activation_model = tf.keras.models.Model(inputs = model.input, outputs = layer_outputs)
for x in range(0,4):
  f1 = activation_model.predict(test_images[FIRST_IMAGE].reshape(1, 28, 28, 1))[x]
  axarr[0,x].imshow(f1[0, : , :, CONVOLUTION_NUMBER], cmap='inferno')
  axarr[0,x].grid(False)
  f2 = activation_model.predict(test_images[SECOND_IMAGE].reshape(1, 28, 28, 1))[x]
  axarr[1,x].imshow(f2[0, : , :, CONVOLUTION_NUMBER], cmap='inferno')
  axarr[1,x].grid(False)
  f3 = activation_model.predict(test_images[THIRD_IMAGE].reshape(1, 28, 28, 1))[x]
  axarr[2,x].imshow(f3[0, : , :, CONVOLUTION_NUMBER], cmap='inferno')
  axarr[2,x].grid(False)
plt.show()
```

Examples using cv2

```
import cv2
import numpy as np
from scipy import misc
import matplotlib.pyplot as plt

i = misc.ascent()
row, cols = plt.subplots(1, 3)
plt.grid(False)
plt.gray()
cols[0].imshow(i)

i_copy = i.copy()
size_x = i_copy.shape[0]
size_y = i_copy.shape[1]

filter = [[0, 1, 0], [1, -4, 1], [0, 1, 0]]
```

```
filter = [[-1, -1, -2], [0, 0, 0], [1, 2, 1]] # Vertical lines
filter = [[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]] # Horizontal lines
weight = 1

#------------------------------------------------------------------------#
# This part handles the Convolution
for x in range(1, size_x-1):
    for y in range(1, size_y-1):
        convolution = 0
        convolution = convolution + (i[x-1, y-1] * filter[0][0])
        convolution = convolution + (i[x, y-1] * filter[0][1])
        convolution = convolution + (i[x+1, y-1] * filter[0][2])
        convolution = convolution + (i[x-1, y] * filter[1][0])
        convolution = convolution + (i[x, y] * filter[1][1])
        convolution = convolution + (i[x+1, y] * filter[1][2])
        convolution = convolution + (i[x-1, y+1] * filter[2][0])
        convolution = convolution + (i[x, y+1] * filter[2][1])
        convolution = convolution + (i[x+1, y+1] * filter[2][2])
        convolution *= weight
        if convolution < 0:
            convolution = 0
        if convolution > 255:
            convolution = 255
        i_copy[x][y] = convolution
#------------------------------------------------------------------------#

cols[1].imshow(i_copy)

#------------------------------------------------------------------------#
# This part handles the pooling of the image
new_x = int(size_x/2)
new_y = int(size_y/2)
new_image = np.zeros((new_x, new_y))
for x in range(0, size_x, 2):
    for y in range(0, size_y, 2):
        pixel = []
        pixel.append(i_copy[x, y])
        pixel.append(i_copy[x+1, y])
        pixel.append(i_copy[x, y+1])
        pixel.append(i_copy[x+1, y+1])
        pixel.sort(reverse=True)
        new_image[int(x/2), int(y/2)] = pixel[0]
#------------------------------------------------------------------------#

cols[2].imshow(new_image)

plt.show()
```

Weekly test

```
import tensorflow as tf
from os import path, getcwd, chdir
import matplotlib.pyplot as plt

# DO NOT CHANGE THE LINE BELOW. If you are developing in a local
# environment, then grab mnist.npz from the Coursera Jupyter Notebook
# and place it inside a local folder and edit the path to that locationE

# GRADED FUNCTION: train_mnist_conv

class myCallback(tf.keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs={}):
    if(logs.get('accuracy')>0.998):
```

```
      print("\nReached 99.8% accuracy so cancelling training!")
      self.model.stop_training = True

def train_mnist_conv():
    # Please write your code only where you are indicated.
    # please do not remove model fitting inline comments.

    # YOUR CODE STARTS HERE
    # YOUR CODE ENDS HERE
    mnist = tf.keras.datasets.mnist
    (training_images, training_labels), (test_images, test_labels) = mnist.load_data()
    # YOUR CODE STARTS HERE
    callback = myCallback()
    training_images=training_images.reshape(60000, 28, 28, 1)
    training_images=training_images / 255.0

    test_images = test_images.reshape(10000, 28, 28, 1)
    test_images=test_images/255.0
    # YOUR CODE ENDS HERE

    model = tf.keras.models.Sequential([
        tf.keras.layers.Conv2D(64, (3,3), activation='relu', input_shape=(28, 28, 1)),
        tf.keras.layers.MaxPool2D(2, 2),
        tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
        tf.keras.layers.MaxPool2D(2, 2),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(128, activation='relu'),
        tf.keras.layers.Dense(10, activation='softmax')
    ])

    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    # model fitting
    history = model.fit(training_images, training_labels, epochs=20, callbacks=[callback])
    # model fitting
    return history.epoch, history.history['accuracy'][-1]

_, _ = train_mnist_conv()
```

Practice exercise

```
import tensorflow as tf

class CallBack(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, log={}):
        if log.get('accuracy') > 0.998:
            print("Reached 99.8% accuracy so cancelling training!")
            self.model.stop_training = True

callback = CallBack()

mnist = tf.keras.datasets.mnist
(training_images, training_labels), (test_images, test_labels) = mnist.load_data()

training_images = training_images.reshape(60000, 28, 28, 1)
training_images = training_images / 255.0

test_images = test_images.reshape(10000, 28, 28, 1)
test_images = test_images / 255.0

model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(64, (3,3), activation='relu', input_shape=(28, 28, 1)),
```

```
    tf.keras.layers.MaxPool2D(2, 2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPool2D(2, 2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.fit(training_images, training_labels, epochs=20, callbacks=[callback])
```