Improving Computer Vision Accuracy using Convolutions

In the previous lessons you saw how to do fashion recognition using a Deep Neural Network (DNN the shape of the data), the output layer (in the shape of the desired output) and a hidden layer. You sized of hidden layer, number of training epochs etc on the final accuracy.

For convenience, here's the entire code again. Run it and take a note of the test accuracy that is pr

```
import tensorflow as tf
mnist = tf.keras.datasets.fashion_mnist
(training_images, training_labels), (test_images, test_labels) = mnist.load_da
training_images=training_images / 255.0
test_images=test_images / 255.0
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(128, activation=tf.nn.relu),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metric
model.fit(training_images, training_labels, epochs=5)

test_loss = model.evaluate(test_images, test_labels)
```

The default version of TensorFlow in Colab will soon switch to TensorFlow 2.x.
We recommend you upgrade now or ensure your notebook will continue to use TensorFlow 1.x via the `%tensorflow_version 1.x` magic: more info.

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datas
32768/29515 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datas
26427392/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datas
8192/5148 [===============================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datas
4423680/4422102 [==============================] - 0s 0us/step
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
Train on 60000 samples
Epoch 1/5
60000/60000 [==============================] - 4s 66us/sample - loss: 0.4975 -
Epoch 2/5
60000/60000 [==============================] - 4s 65us/sample - loss: 0.3740 -
Epoch 3/5
60000/60000 [==============================] - 4s 63us/sample - loss: 0.3341 -
Epoch 4/5
60000/60000 [==============================] - 4s 62us/sample - loss: 0.3105 -
Epoch 5/5
60000/60000 [==============================] - 4s 62us/sample - loss: 0.2937 -
10000/10000 [==============================] - 0s 31us/sample - loss: 0.3483 -
```

Your accuracy is probably about 89% on training and 87% on validation...not bad...But how do you something called Convolutions. I'm not going to details on Convolutions here, but the ultimate con the image to focus on specific, distinct, details.

If you've ever done image processing using a filter (like this: https://en.wikipedia.org/wiki/Kernel_( look very familiar.

In short, you take an array (usually 3x3 or 5x5) and pass it over the image. By changing the underly matrix, you can do things like edge detection. So, for example, if you look at the above link, you'll se where the middle cell is 8, and all of its neighbors are -1. In this case, for each pixel, you would mu each neighbor. Do this for every pixel, and you'll end up with a new image that has the edges enhar

This is perfect for computer vision, because often it's features that can get highlighted like this tha amount of information needed is then much less...because you'll just train on the highlighted featu

That's the concept of Convolutional Neural Networks. Add some layers to do convolution before yo information going to the dense layers is more focussed, and possibly more accurate.

Run the below code -- this is the same neural network as earlier, but this time with Convolutional la the impact on the accuracy:

```
import tensorflow as tf
print(tf.__version__)
mnist = tf.keras.datasets.fashion_mnist
(training_images, training_labels), (test_images, test_labels) = mnist.load_da
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', metric
model.summary()
model.fit(training_images, training_labels, epochs=5)
test_loss = model.evaluate(test_images, test_labels)

```

```
1.15.0
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 64)        640
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1600)              0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               204928
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1290
=================================================================
Total params: 243,786
Trainable params: 243,786
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples
Epoch 1/5
60000/60000 [==============================] - 83s 1ms/sample - loss: 0.4430 -
Epoch 2/5
60000/60000 [==============================] - 82s 1ms/sample - loss: 0.2968 -
Epoch 3/5
60000/60000 [==============================] - 82s 1ms/sample - loss: 0.2511 -
Epoch 4/5
60000/60000 [==============================] - 81s 1ms/sample - loss: 0.2195 -
Epoch 5/5
60000/60000 [==============================] - 81s 1ms/sample - loss: 0.1941 -
10000/10000 [==============================] - 4s 388us/sample - loss: 0.2652
```

It's likely gone up to about 93% on the training data and 91% on the validation data.

That's significant, and a step in the right direction!

Try running it for more epochs -- say about 20, and explore the results! But while the results might actually go down, due to something called 'overfitting' which will be discussed later.

(In a nutshell, 'overfitting' occurs when the network learns the data from the training set really well, as a result is less effective at seeing *other* data. For example, if all your life you only saw red shoes very good at identifying it, but blue suade shoes might confuse you...and you know you should nev

Then, look at the code again, and see, step by step how the Convolutions were built:

Step 1 is to gather the data. You'll notice that there's a bit of a change here in that the training data first convolution expects a single tensor containing everything, so instead of 60,000 28x28x1 item 60,000x28x28x1, and the same for the test images. If you don't do this, you'll get an error when tra the shape.

```
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
```

Next is to define your model. Now instead of the input layer at the top, you're going to add a Convo

1. The number of convolutions you want to generate. Purely arbitrary, but good to start with something in the
2. The size of the Convolution, in this case a 3x3 grid
3. The activation function to use -- in this case we'll use relu, which you might recall is the equivalent of retur
4. In the first layer, the shape of the input data.

You'll follow the Convolution with a MaxPooling layer which is then designed to compress the imag features that were highlighted by the convlution. By specifying (2,2) for the MaxPooling, the effect going into too much detail here, the idea is that it creates a 2x2 array of pixels, and picks the bigge this across the image, and in so doing halves the number of horizontal, and halves the number of v by 25%.

You can call model.summary() to see the size and shape of the network, and you'll notice that afte reduced in this way.

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(28, 28, 1)),
  tf.keras.layers.MaxPooling2D(2, 2),
```

Add another convolution

```
  tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
  tf.keras.layers.MaxPooling2D(2,2)
```

Now flatten the output. After this you'll just have the same DNN structure as the non convolutional

```
  tf.keras.layers.Flatten(),
```

The same 128 dense layers, and 10 output layers as in the pre-convolution example:

```
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(10, activation='softmax')
```

```
])
```

Now compile the model, call the fit method to do the training, and evaluate the loss and accuracy f

```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy
model.fit(training_images, training_labels, epochs=5)
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(test_acc)
```

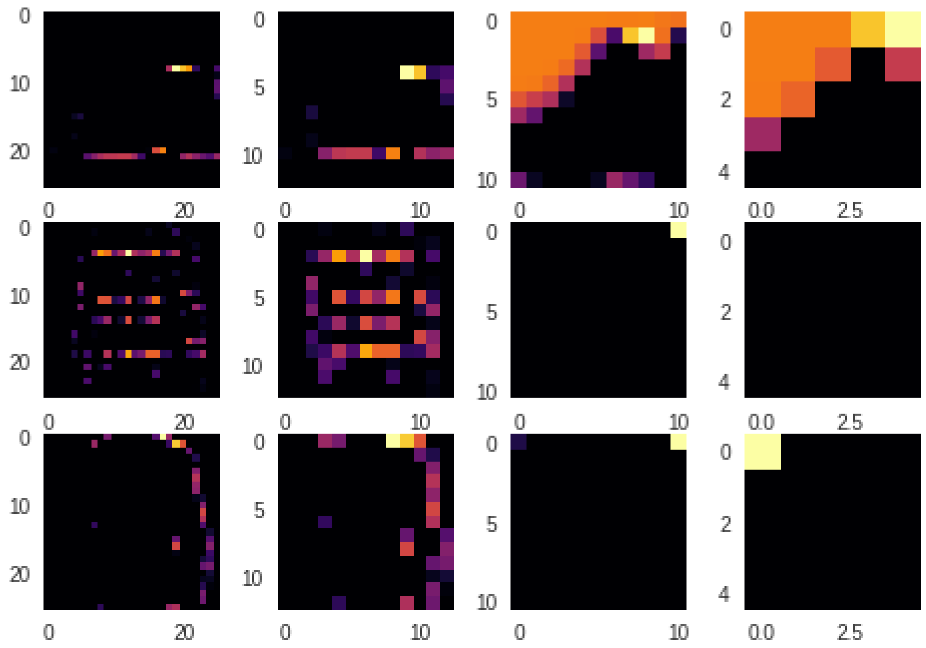
Visualizing the Convolutions and Pooling

This code will show us the convolutions graphically. The print (test_labels[;100]) shows us the first that the ones at index 0, index 23 and index 28 are all the same value (9). They're all shoes. Let's ta convolution on each, and you'll begin to see common features between them emerge. Now, when t with a lot less, and it's perhaps finding a commonality between shoes based on this convolution/p

```
print(test_labels[:100])
```

```
[9 2 1 1 6 1 4 6 5 7 4 5 7 3 4 1 2 4 8 0 2 5 7 9 1 4 6 0 9 3 8 8 3 3 8 0 7
 5 7 9 6 1 3 7 6 7 2 1 2 2 4 4 5 8 2 2 8 4 8 0 7 7 8 5 1 1 2 3 9 8 7 0 2 6
 2 3 1 2 8 4 1 8 5 9 5 0 3 2 0 6 5 3 6 7 1 8 0 1 4 2]
```

```
import matplotlib.pyplot as plt
f, axarr = plt.subplots(3,4)
FIRST_IMAGE=
SECOND_IMAGE=7
THIRD_IMAGE=26
CONVOLUTION_NUMBER = 1
from tensorflow.keras import models
layer_outputs = [layer.output for layer in model.layers]
activation_model = tf.keras.models.Model(inputs = model.input, outputs = layer
for x in range(0,4):
  f1 = activation_model.predict(test_images[FIRST_IMAGE].reshape(1, 28, 28, 1)
  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
  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)
  axarr[2,x].imshow(f3[0, : , :, CONVOLUTION_NUMBER], cmap='inferno')
  axarr[2,x].grid(False)
```



EXERCISES

1. Try editing the convolutions. Change the 32s to either 16 or 64. What impact will this have or
2. Remove the final Convolution. What impact will this have on accuracy or training time?
3. How about adding more Convolutions? What impact do you think this will have? Experiment
4. Remove all Convolutions but the first. What impact do you think this will have? Experiment w
5. In the previous lesson you implemented a callback to check on the loss function and to canc you can implement that here!

```
import tensorflow as tf
print(tf.__version__)
mnist = tf.keras.datasets.mnist
(training_images, training_labels), (test_images, test_labels) = mnist.load_da
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(32, (3,3), activation='relu', input_shape=(28, 28, 1)
  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', metri
model.fit(training_images, training_labels, epochs=10)
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(test_acc)
```

--NORMAL--

```
1.12.0
Epoch 1/10
60000/60000==============================] - 6s 104us/sample - loss: 0.1510 -
Epoch 2/10
60000/60000==============================] - 5s 79us/sample - loss: 0.0512 - a
Epoch 3/10
60000/60000==============================] - 5s 77us/sample - loss: 0.0319 - a
Epoch 4/10
60000/60000==============================] - 5s 78us/sample - loss: 0.0209 - a
Epoch 5/10
60000/60000==============================] - 5s 78us/sample - loss: 0.0136 - a
Epoch 6/10
60000/60000==============================] - 5s 78us/sample - loss: 0.0111 - a
Epoch 7/10
60000/60000==============================] - 5s 79us/sample - loss: 0.0076 - a
Epoch 8/10
60000/60000==============================] - 5s 78us/sample - loss: 0.0052 - a
Epoch 9/10
60000/60000==============================] - 5s 81us/sample - loss: 0.0046 - a
Epoch 10/10
60000/60000==============================] - 5s 81us/sample - loss: 0.0053 - a
10000/10000==============================] - 1s 53us/sample - loss: 0.0583 - a
0.9873
```