Stoneburner, Kurt

- ## DSC 650 - Week 06
- ## Assignment 6.1 - ConvNet Model that classifies images in the MNIST digital dataset.

This work is essentially copy/pasted from the book's code. I spent a fair bit of time getting my head wrapped around conv2d and Max Pooling. I added comments to clarify my current understanding and help with future reference.

In [1]:
```
#//*** If this code works, then Directml and GPU operations are worki
import tensorflow.compat.v1 as tf

tf.enable_eager_execution(tf.ConfigProto(log_device_placement=True))

print(tf.add([1.0, 2.0], [3.0, 4.0]))
```

```
Executing op Add in device /job:localhost/replica:0/task:0/device:DML:
0
tf.Tensor([4. 6.], shape=(2,), dtype=float32)
```

Reference: https://keras.io/api/datasets/mnist/ (https://keras.io/api/datasets/mnist/)

Max-Pooling Explained: https://analyticsindiamag.com/max-pooling-in-convolutional-neural-network-and-its-features/ (https://analyticsindiamag.com/max-pooling-in-convolutional-neural-network-and-its-features/)

Conv2D Official Documentation: https://keras.io/api/layers/convolution_layers/convolution2d/ (https://keras.io/api/layers/convolution_layers/convolution2d/)

In [8]:
```
import os
import sys
# //*** Imports and Load Data
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

#//*** Use the whole window in the IPYNB editor
from IPython.core.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))

#//*** Maximize columns and rows displayed by pandas
pd.set_option('display.max_rows', 100)
pd.set_option('display.max_columns', None)

import tensorflow as tf

from tensorflow import keras
from tensorflow.keras import layers
```

Load the MNIST dataset.

This is a dataset of 60,000 28x28 grayscale images of the 10 digits, along with a test set of 10,000 images.

Returns

```
Tuple of NumPy arrays: (x_train, y_train), (x_test, y_test).
```

x_train: uint8 NumPy array of grayscale image data with shapes (60000, 28, 28), containing the training data. Pixel values range from 0 to 255.

y_train: uint8 NumPy array of digit labels (integers in range 0-9) with shape (60000,) for the training data.

x_test: uint8 NumPy array of grayscale image data with shapes (10000, 28, 28), containing the test data. Pixel values range from 0 to 255.

y_test: uint8 NumPy array of digit labels (integers in range 0-9) with shape (10000,) for the test data.

In [10]:
```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_d
assert x_train.shape == (60000, 28, 28)
assert x_test.shape == (10000, 28, 28)
assert y_train.shape == (60000,)
assert y_test.shape == (10000,)
```

In [11]:
```


#//*** Inputs reflects the shape of each individual piece of data.
#//*** The MNIST is a 28x28 single channel image.
#//*** The third is 1 channel. This is a greyscale image, therefore i
#//*** See Link above for further explanation
inputs = keras.Input(shape=(28, 28, 1))

#//*** Conv2D: Filters defines the number of tensors at the layer. Ke
#//***         Conv2D reduces the image size by (filter_size - 1) in
#//*** MaxPooling2D: Is a form of feature reduction. In this case, Th
#//***               pool-size value (2x2) is kept. With a pool value
#//***               by 75%.
#//***               At each stage of max pooling, the image gets sma
#//***               for relationships between the reduced features.

x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(input
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(10, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)

```

```
model.summary()
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 28, 28, 1)]       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 3, 3, 128)         73856
_________________________________________________________________
flatten_1 (Flatten)          (None, 1152)              0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                11530
=================================================================
Total params: 104,202
Trainable params: 104,202
Non-trainable params: 0
_________________________________________________________________
```

In [14]:

```
x_train = x_train.reshape((60000, 28, 28, 1))
x_train = x_train.astype("float32") / 255
x_test = x_test.reshape((10000, 28, 28, 1))
x_test = x_test.astype("float32") / 255
model.compile(optimizer="rmsprop",
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"])
```

```
Epoch 1/5
938/938 [==============================] - 21s 21ms/step - loss: 1.331
2 - accuracy: 0.5624
Epoch 2/5
938/938 [==============================] - 20s 21ms/step - loss: 0.275
9 - accuracy: 0.9167
Epoch 3/5
938/938 [==============================] - 20s 21ms/step - loss: 0.133
6 - accuracy: 0.9598
Epoch 4/5
938/938 [==============================] - 20s 21ms/step - loss: 0.088
3 - accuracy: 0.9736
Epoch 5/5
938/938 [==============================] - 20s 21ms/step - loss: 0.068
1 - accuracy: 0.9791
```

Out[14]:

```
<tensorflow.python.keras.callbacks.History at 0x1cf87a3e160>
```

In [15]:

```
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"Test accuracy: {test_acc:.3f}")


```

```
313/313 [==============================] - 1s 4ms/step - loss: 0.0557
- accuracy: 0.9826
Test accuracy: 0.983
```