Course 1 - Introduction to TensorFlow for AI, ML & DL

> // Notes by Om

Week 1

```
model = tf.keras.Sequential([
  tf.keras.Input(shape=(1,)),
  tf.keras.layers.Dense(units=1)
])

model.compile(optimizer='sgd', loss='mean_
xs = np.array([-1.0, 0.0, 1.0, 2.0, 3.0, 4.0], dty
ys = np.array([-3.0, -1.0, 1.0, 3.0, 5.0, 7.0], dty

model.fit(x=xs, y=ys, epochs=500)
```

This is the “Hello World” for tensorflow and deep learning
The
`Dense` layer is the most basic layer - A layer of connected neurons
Here we have a single layer with a single neuron in it

Optimizers and loss

The loss function calculates the closeness of the predicted values to the dataset. The loss function measures how good or bad the guess was and then passes this to the optimizer which then refines the model.

Model training

The compile function tells what to use on each step. The training step tells the model which training data to use & the number of epochs which is the number of training cycles.

In every training cycle, the loss is calculated and then the optimizer optimizes the model

Week 2 - Computer Vision with TensorFlow

```
# Computer vision neural network
model = tf.keras.Sequential([
   tf.keras.Input(shape=(28, 28)),
   tf.keras.layers.Flatten(),
   tf.keras.layers.Dense(units=128, activation=
   tf.keras.layers.Dense(units=10, activation=
])

model.summary()
```

The Input layer's shape is (28, 28) because that is the size of the images in the MNIST dataset

Flatten layer takes the (28, 28) images and turns it into simple linear array

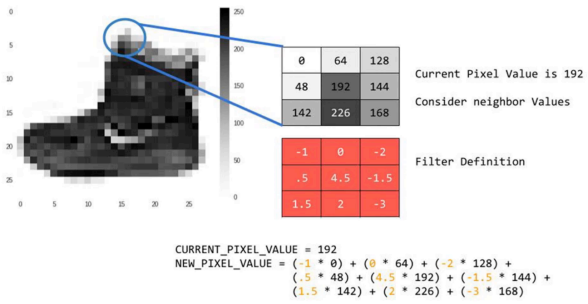
```
Model: "sequential"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ flatten (Flatten)               │ (None, 784)            │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense (Dense)                   │ (None, 128)            │       100,480 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_1 (Dense)                 │ (None, 10)             │         1,290 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 101,770 (397.54 KB)

 Trainable params: 101,770 (397.54 KB)

 Non-trainable params: 0 (0.00 B)
```

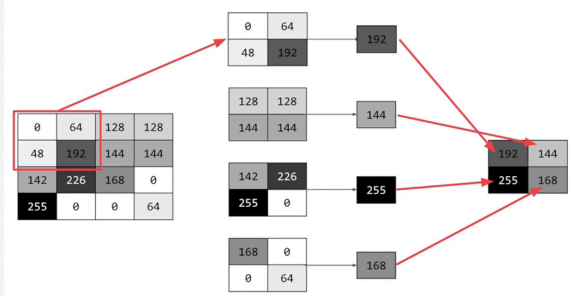
Week 3 - Enhancing Vision with Convolutional Neural Networks

When we don't use CNNs the model looks at the value of every pixel and then makes predictions. But with CNNs, the model essentially focuses on detecting specific features like the handle of a bag or the shoelace for a shoe. Once these features are found,



Convolution filters and pooling

Some convolutions enhance the image in such a way that it makes some features pop out.



Pooling uses the current pixel and its adjacent pixels. Then from all these, the highest value is extracted and kept to form another final pixel.

This reduces the size of the image

For instance:

The 16 pixels on the left were converted to 4 pixels on the right

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

In the code on the left:

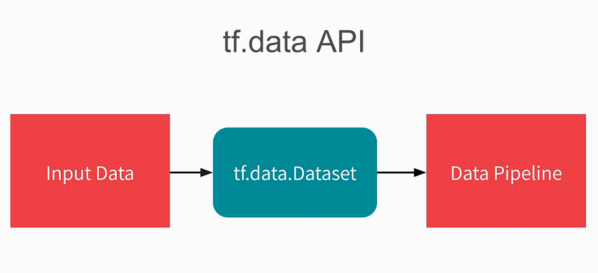
1. The (28, 28, 1) stands for input image dimensions with 1 for colors
2. The Conv2D layer has 64 filters with (3,3) size as shown in the illustrations above
3. The MaxPooling2D layer takes the maximum value from a set of pixels. The (2,2) signifies that the max value would be taken from 4 pixels. So max of the 4 pixels as shown in the illustration.
4. Other layers like Flatten(), Dense layers remain the same

Reshape images which are of format `np.array` :

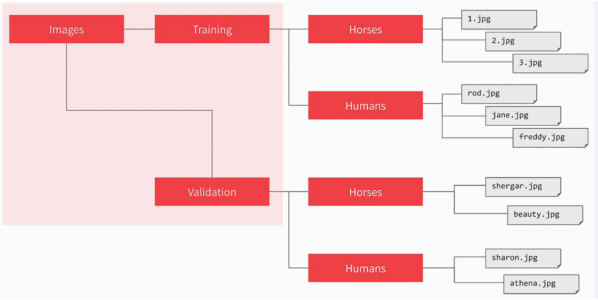
```
images = images.reshape(-1, 28, 28, 1)
```

Week 4

tf.data API



We have a input data source and we feed it to the `tf.data.dataset` object. This can then be used to perform certain actions on the data




```
train_dataset = tf.keras.utils.image_dataset_fr
    TRAIN_DIR,
    image_size=(300,300),
    batch_size=128,
    label_mode='binary'
)
```

`tf.keras.utils.image_dataset_from_dir`

- If we point the above function at the training directory, then it will parse and consider horses and humans as the labels. It will label all the. images in these directories with the corresponding labels
- This also resizes the images if they aren’t in the same size
- The output would be a `tf.data.dataset` object which helps preprocess the data through various method calls which are together call the ***data pipeline***.
- ***Some of the important functions from the data pipeline are***
    - cache(): Helps to store some data in the cache memory for faster retrieval
    - shuffle(buffer_size): Shuffles the data with a buffer to keep

images

- prefetch(): Prefetches the next batch of data while training the model on the previous one