Hello all ....

> Hope you are doing good....... I am Jayati Vijaywargiya, and here I bring a small tutorial to start working on Convolution Neural Network using tensorflow. In this excercise we will

- Load and analyze MNIST dataset
- Divide dataset in training, testing and validation dataset
- Prepare a basic convolutional neural network model for classification
- Compile the model
- Display the model summary
- Use cross-entropy loss and accuracy metrics for the model prepared
- Train the network
- Validate the network using validate set and show the results

So lets get started

+ Code + Text

```
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
import tensorflow_datasets as tfds
from tensorflow.keras import datasets,layers, models
from sklearn.model_selection import train_test_split
```

In the above step we imported all the necessary libraries, we would be using

Now, we will load the data set,

> Before that, let me brief a little bit about dataset.
>
> > The MNIST dataset is an acronym that stands for the Modified National Institute of Standards and Technology dataset. It is a dataset of 60,000 small square 28×28 pixel grayscale images of handwritten single digits between 0 and 9.
> >
> > > It is loaded in 2 sets, training set of 60000 images and testing set of 10000 images

```
(Itrain_images, Itrain_labels), (test_images, test_labels) = datasets.mnist.load_data()
```

```
np.shape(train_images_set1)
```

```
(60000, 28, 28, 1)
```

```
train_images, val_images , train_labels, val_labels = train_test_split(Itrain_images, Itra
                                                    train_size=0.8,
                                                    random_state=42)
```

In the above code, we have divided the training and validation data from the training data set.

```
np.shape(train_images)
```

```
(48000, 28, 28)
```

```
np.shape(val_images)
```

```
(12000, 28, 28)
```

Now, we will see the size of each training data, the class labels and the number of classes in the training dataset

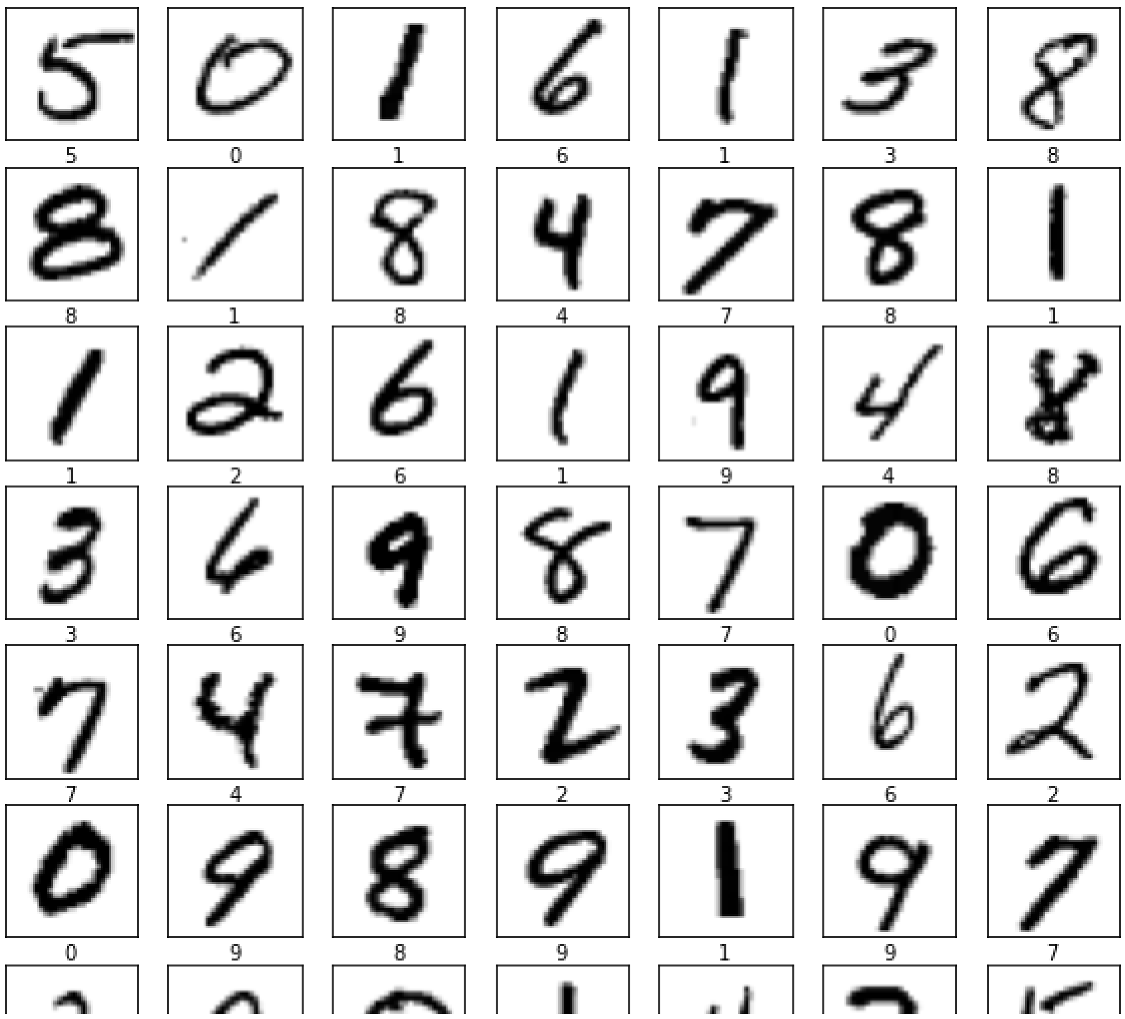
```
print(' Image size = ', np.shape(train_images[1]))
print(' Labels =' , np.unique(train_labels))
print('Number of classes', np.size(np.unique(train_labels)))
```

```
 Image size =  (28, 28)
 Labels = [0 1 2 3 4 5 6 7 8 9]
Number of classes 10
```

In this dataset, we have each data as a 28*28 image, and we have 10 classes, labelled as 0,1,2,3,4,5,6,7,8,9

Now, we will plot a few training data images

```
plt.figure(figsize=(10,10))
for i in range(49):
    plt.subplot(7,7,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.xlabel(train_labels[i])
plt.show()
```



By now, you might be well aware about the dataset

> Let's go to Convolution Neural Network Modelling
>
> > If you dont know much about its theory part, Please spend 5 minutes and check this video https://www.youtube.com/watch?v=6Y4BEwAo1i8

```
from keras.utils import to_categorical
train_images=train_images.reshape((train_images.shape[0], 28, 28, 1))
test_images=test_images.reshape((test_images.shape[0], 28, 28, 1))
val_images=val_images.reshape((val_images.shape[0], 28, 28, 1))
train_images_set1=train_images_set1.reshape((train_images_set1.shape[0], 28, 28, 1))
```

```
model = models.Sequential()
model.add(layers.Conv2D(32, (5, 5), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(16, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(40, activation='relu'))
model.add(layers.Dense(20, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

In the above code, we created a Convolution Neural Network, explaining that in detail..........Be patient if you want to understand.

The modeled CNN has following architechture

1. layer 1- Convolution layer (32 filters, 3*3 kernel size and ReLU activation funtion) # for more detail about inbuilt function used - https://keras.io/api/layers/convolution_layers/convolution2d/
2. layer 2- Convolution layer (32 filters, 3*3 kernel and ReLU activation funtion) # for more detail about inbuilt function used - https://keras.io/api/layers/convolution_layers/convolution2d/
3. Layer 3- Pooling layer of pool size 2*2, for spatial dimensionality reduction # for more detail about inbuilt function used https://keras.io/api/layers/pooling_layers/max_pooling2d/
4. Layer 4- Convolution layer(16 filters, 3*3 kernel and ReLU activation funtion) # for more detail about inbuilt function used - https://keras.io/api/layers/convolution_layers/convolution2d/
5. Layer 5- Pooling layer of pool size 2*2, for spatial dimensionality reduction # for more detail about inbuilt function used https://keras.io/api/layers/pooling_layers/max_pooling2d/
6. Layer 6 is a flattening layer, it is used to flatten the input. For example, if flatten is applied to layer having input shape as (batch_size, 2,2), then the output shape of the layer will be (batch_size, 4) # for more detail about inbuilt function used https://www.tutorialspoint.com/keras/keras_flatten_layers.htm
7. Layer 7 is a fully connected layer or Dense layer. It uses ReLU activation function. It is like a regular densely-connected NN layer, In the abobe model the output of this layer will have 100 unique outputs or 100 classes # for more detail about inbuilt function used https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense
8. Layer 8 is also a fully connected layer or Dense layer. It uses softmax activation function. It is like a regular densely-connected NN layer, In the abobe model the output of this layer will have 10 unique outputs or 10 classes # for more detail about inbuilt function used https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense

The above model explained was first tried, but it did not yeild good accuracy

so, methods like,

Increase hidden Layers. ... Change Activation function. ... Change Activation function in Output layer. ... Increase number of neurons. ..

were applied

This was all about the simple Convolution Neural Network modelled in the above code.

The summary of the new model created is here,

```
model.summary()
```

```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_29 (Conv2D)           (None, 24, 24, 32)        832       
_________________________________________________________________
conv2d_30 (Conv2D)           (None, 22, 22, 32)        9248      
_________________________________________________________________
conv2d_31 (Conv2D)           (None, 20, 20, 32)        9248      
_________________________________________________________________
max_pooling2d_24 (MaxPooling (None, 10, 10, 32)        0         
_________________________________________________________________
conv2d_32 (Conv2D)           (None, 8, 8, 16)          4624      
_________________________________________________________________
max_pooling2d_25 (MaxPooling (None, 4, 4, 16)          0         
_________________________________________________________________
flatten_12 (Flatten)         (None, 256)               0         
_________________________________________________________________
dense_27 (Dense)             (None, 40)                10280     
_________________________________________________________________
dense_28 (Dense)             (None, 20)                820       
_________________________________________________________________
dense_29 (Dense)             (None, 10)                210       
=================================================================
Total params: 35,262
Trainable params: 35,262
Non-trainable params: 0
_________________________________________________________________
```

After model creation, model needs to be compiled. Here, Adam optimization is used.

Adam optimization

> The Adam optimization algorithm is an extension to stochastic gradient descent that has recently seen broader adoption for deep learning applications in computer vision and natural language processing.

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Now we will fit the model, using traing data and validation data, using 20 epochs or iterations

Note: the below step will take time to execute

```
model_1= model.fit(train_images, train_labels, epochs=50,validation_data=(val_images, val_
1500/1500 [==============================] - 5s 3ms/step - loss: 1.8979 - accuracy: 0
Epoch 15/50
1500/1500 [==============================] - 5s 3ms/step - loss: 1.9511 - accuracy: 0
Epoch 16/50
1500/1500 [==============================] - 5s 3ms/step - loss: 1.9395 - accuracy: 0
Epoch 17/50
1500/1500 [==============================] - 5s 3ms/step - loss: 1.9747 - accuracy: 0
Epoch 18/50
1500/1500 [==============================] - 5s 3ms/step - loss: 1.9715 - accuracy: 0
Epoch 19/50
1500/1500 [==============================] - 5s 3ms/step - loss: 1.8919 - accuracy: 0
Epoch 20/50
1500/1500 [==============================] - 5s 3ms/step - loss: 1.9529 - accuracy: 0
Epoch 21/50
1500/1500 [==============================] - 5s 3ms/step - loss: 1.9932 - accuracy: 0
Epoch 22/50
1500/1500 [==============================] - 5s 3ms/step - loss: 1.9382 - accuracy: 0
Epoch 23/50
1500/1500 [==============================] - 5s 3ms/step - loss: 1.8842 - accuracy: 0
Epoch 24/50
1500/1500 [==============================] - 5s 3ms/step - loss: 1.9352 - accuracy: 0
Epoch 25/50
1500/1500 [==============================] - 5s 3ms/step - loss: 2.0391 - accuracy: 0
Epoch 26/50
1500/1500 [==============================] - 5s 3ms/step - loss: 2.0231 - accuracy: 0
Epoch 27/50
1500/1500 [==============================] - 5s 3ms/step - loss: 2.0630 - accuracy: 0
Epoch 28/50
1500/1500 [==============================] - 5s 3ms/step - loss: 2.0915 - accuracy: 0
Epoch 29/50
1500/1500 [==============================] - 5s 3ms/step - loss: 2.0053 - accuracy: 0
Epoch 30/50
1500/1500 [==============================] - 5s 3ms/step - loss: 1.9635 - accuracy: 0
Epoch 31/50
1500/1500 [==============================] - 5s 3ms/step - loss: 1.9714 - accuracy: 0
Epoch 32/50
1500/1500 [==============================] - 5s 3ms/step - loss: 1.9714 - accuracy: 0
Epoch 33/50
1500/1500 [==============================] - 5s 3ms/step - loss: 1.9714 - accuracy: 0
Epoch 34/50
1500/1500 [==============================] - 5s 3ms/step - loss: 1.9714 - accuracy: 0
Epoch 35/50
1500/1500 [==============================] - 5s 3ms/step - loss: 1.9512 - accuracy: 0
Epoch 36/50
1500/1500 [==============================] - 5s 3ms/step - loss: 1.9258 - accuracy: 0
Epoch 37/50
1500/1500 [==============================] - 5s 3ms/step - loss: 2.0571 - accuracy: 0
Epoch 38/50
1500/1500 [==============================] - 5s 3ms/step - loss: 2.0683 - accuracy: 0
Epoch 39/50
1500/1500 [==============================] - 5s 3ms/step - loss: 2.0289 - accuracy: 0
Epoch 40/50
```

From the above we got the intuition that the first 20 or 15 epochs are sufficient and yeild better result

now we will again create the another model and train it for 10 epochs

```
1500/1500 [==============================] - 5s 3ms/step - loss: 2.2333 - accuracy: 0
```

```
model2 = models.Sequential()
model2.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model2.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model2.add(layers.MaxPooling2D((2, 2)))
model2.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model2.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model2.add(layers.MaxPooling2D((2, 2)))
model2.add(layers.Conv2D(16, (3, 3), activation='relu'))
model2.add(layers.MaxPooling2D((2, 2)))
model2.add(layers.Flatten())
model2.add(layers.Dense(40, activation='softmax'))
model2.add(layers.Dense(20, activation='softmax'))
model2.add(layers.Dense(10, activation='softmax'))
model2.summary()
```

```
Model: "sequential_16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_41 (Conv2D)           (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_42 (Conv2D)           (None, 24, 24, 32)        9248
_________________________________________________________________
max_pooling2d_32 (MaxPooling (None, 12, 12, 32)        0
_________________________________________________________________
conv2d_43 (Conv2D)           (None, 10, 10, 32)        9248
_________________________________________________________________
conv2d_44 (Conv2D)           (None, 8, 8, 32)          9248
_________________________________________________________________
max_pooling2d_33 (MaxPooling (None, 4, 4, 32)          0
_________________________________________________________________
conv2d_45 (Conv2D)           (None, 2, 2, 16)          4624
_________________________________________________________________
max_pooling2d_34 (MaxPooling (None, 1, 1, 16)          0
_________________________________________________________________
flatten_16 (Flatten)         (None, 16)                0
_________________________________________________________________
dense_36 (Dense)             (None, 40)                680
_________________________________________________________________
dense_37 (Dense)             (None, 20)                820
_________________________________________________________________
dense_38 (Dense)             (None, 10)                210
=================================================================
Total params: 34,398
Trainable params: 34,398
Non-trainable params: 0
_________________________________________________________________
```

```
model2.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```
model_2= model2.fit(train_images, train_labels, epochs=10,validation_data=(val_images, val
Epoch 1/10
1500/1500 [==============================] - 5s 3ms/step - loss: 2.2890 - accuracy: 0
Epoch 2/10
1500/1500 [==============================] - 5s 3ms/step - loss: 2.2859 - accuracy: 0
Epoch 3/10
1500/1500 [==============================] - 5s 3ms/step - loss: 2.2485 - accuracy: 0
Epoch 4/10
1500/1500 [==============================] - 5s 3ms/step - loss: 2.1877 - accuracy: 0
Epoch 5/10
1500/1500 [==============================] - 5s 3ms/step - loss: 2.2309 - accuracy: 0
Epoch 6/10
1500/1500 [==============================] - 5s 3ms/step - loss: 2.2886 - accuracy: 0
Epoch 7/10
1500/1500 [==============================] - 5s 3ms/step - loss: 2.2950 - accuracy: 0
Epoch 8/10
1500/1500 [==============================] - 5s 4ms/step - loss: 2.2852 - accuracy: 0
Epoch 9/10
1500/1500 [==============================] - 5s 3ms/step - loss: 2.2833 - accuracy: 0
Epoch 10/10
```

After trying a few models, the model below gave good accuracy.

This is how a simple CNN is implemented

The final model is implemented below and has been validated with validation data

And tested with testing data

```
model3 = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(16, (3,3), activation='relu', input_shape=(28, 28,1)),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])
```

```
model3.summary()
```

```
Model: "sequential_27"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_63 (Conv2D)           (None, 26, 26, 16)        160
_________________________________________________________________
max_pooling2d_49 (MaxPooling (None, 13, 13, 16)        0
```

```
model3.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['acc'])
```

```
_________________________________________________________________
```

```
model_3= model3.fit(train_images, train_labels, epochs=10,validation_data=(val_images, val
```

```
Epoch 1/10
1500/1500 [==============================] - 4s 3ms/step - loss: 0.6047 - acc: 0.9378
Epoch 2/10
1500/1500 [==============================] - 4s 2ms/step - loss: 0.0757 - acc: 0.9773
Epoch 3/10
1500/1500 [==============================] - 4s 2ms/step - loss: 0.0550 - acc: 0.9826
Epoch 4/10
1500/1500 [==============================] - 4s 3ms/step - loss: 0.0494 - acc: 0.9851
Epoch 5/10
1500/1500 [==============================] - 4s 2ms/step - loss: 0.0396 - acc: 0.9883
Epoch 6/10
1500/1500 [==============================] - 4s 3ms/step - loss: 0.0372 - acc: 0.9894
Epoch 7/10
1500/1500 [==============================] - 4s 3ms/step - loss: 0.0312 - acc: 0.9910
Epoch 8/10
1500/1500 [==============================] - 4s 3ms/step - loss: 0.0259 - acc: 0.9926
Epoch 9/10
1500/1500 [==============================] - 4s 3ms/step - loss: 0.0288 - acc: 0.9923
Epoch 10/10
1500/1500 [==============================] - 4s 3ms/step - loss: 0.0243 - acc: 0.9937
```

```
y_test = model3.predict(test_images)
```

```
history = model3.fit(train_images, train_labels, epochs=10,validation_data=(test_images, t
```

```
Epoch 1/10
1500/1500 [==============================] - 4s 3ms/step - loss: 0.0177 - acc: 0.9954
Epoch 2/10
1500/1500 [==============================] - 4s 3ms/step - loss: 0.0227 - acc: 0.9950
Epoch 3/10
1500/1500 [==============================] - 4s 3ms/step - loss: 0.0161 - acc: 0.9963
Epoch 4/10
1500/1500 [==============================] - 4s 3ms/step - loss: 0.0219 - acc: 0.9952
Epoch 5/10
1500/1500 [==============================] - 4s 3ms/step - loss: 0.0149 - acc: 0.9965
Epoch 6/10
1500/1500 [==============================] - 4s 3ms/step - loss: 0.0130 - acc: 0.9972
Epoch 7/10
1500/1500 [==============================] - 4s 3ms/step - loss: 0.0202 - acc: 0.9964
Epoch 8/10
1500/1500 [==============================] - 4s 3ms/step - loss: 0.0136 - acc: 0.9974
Epoch 9/10
1500/1500 [==============================] - 4s 3ms/step - loss: 0.0204 - acc: 0.9963
Epoch 10/10
1500/1500 [==============================] - 4s 3ms/step - loss: 0.0185 - acc: 0.9966
```

```
np.shape(y_test)
```

```
(10000, 10)
```

Thus using model 3 a test accuracy of 97.92% is reached

```
import pandas as pd
results = np.argmax(y_test,axis = 1)

results = pd.Series(results,name="Label")
```

```
np.shape(results)
```

```
(10000,)
```

```
from sklearn.metrics import accuracy_score
accuracy_score(test_labels,results)
```

```
0.9792
```