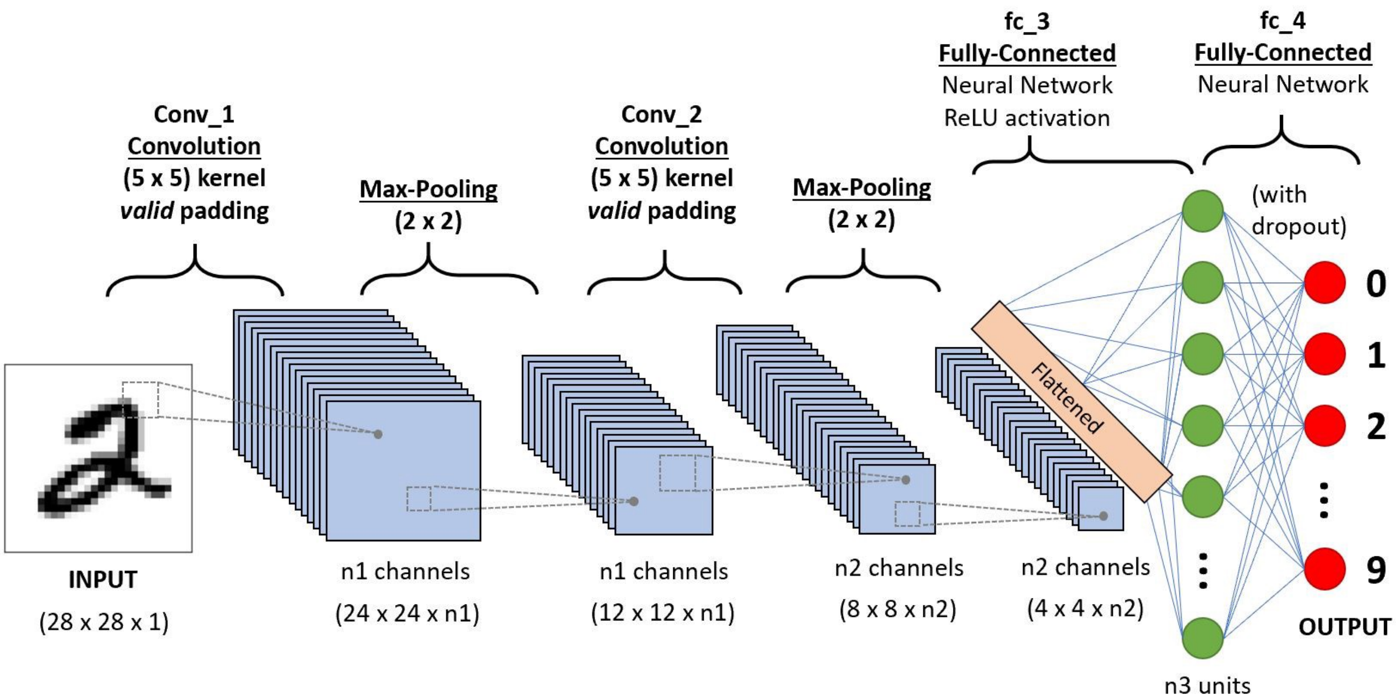
Convolutional Neural Networks

In deep learning, a convolutional neural network is a class of deep neural networks, most commonly applied to analyzing visual imagery.



To learn more about CNN and the various layers of CNN, you can visit [this link.]

Imports

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Loading MNIST Dataset

[MNIST Dataset Wikipedia Link]

```
from tensorflow.keras.datasets import mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()  # Loading dataset
```

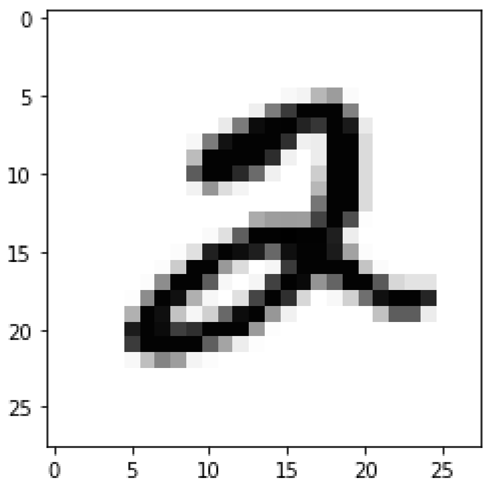
```
# Check shape of dataset
print(x_train.shape)
print(y_train.shape)
```

```
print(y_train.shape)
```

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

```
plt.imshow(x_train[5], cmap='Greys')
```

```
<matplotlib.image.AxesImage at 0x7f73d9b12940>
```




```
y_train[5]
```

```
2
```

One Hot Encoding Dependant Variable (y)

```
from tensorflow.keras.utils import to_categorical
```

```
y_cat_test = to_categorical(y_test, 10)
```

```
y_cat_train = to_categorical(y_train, 10)
```

```
y_cat_test.shape
```

```
(10000, 10)
```

Scaling Data

```
x_train[0].max()
```

```
255
```

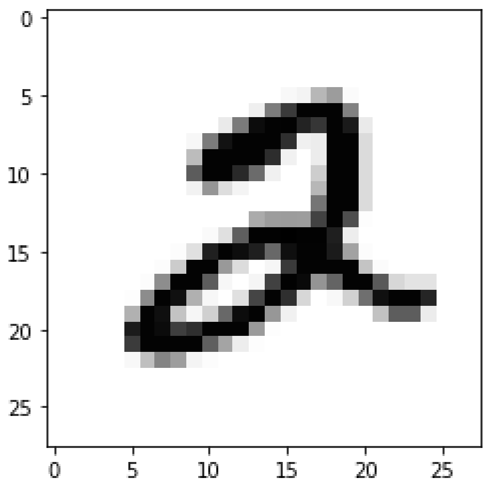
```
# Converting values from [0, 255] range to [0, 1] range
x_train = x_train / 255
x_test = x_test / 255
```

```
x_train[0].max()
```

```
1.0
```

```
plt.imshow(x_train[5], cmap='Greys')
```

```
<matplotlib.image.AxesImage at 0x7f73d28e0f28>
```



Reshaping Data

```
x_test.shape
```

```
(10000, 28, 28)
```

```
x_train = x_train.reshape(60000, 28, 28, 1)
```

```
x_test = x_test.reshape(10000, 28, 28, 1)
```

```
x_test.shape
```

```
(10000, 28, 28, 1)
```

CNN Model

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, Flatten
```

```
from tensorflow.keras.callbacks import EarlyStopping
```

```
model = Sequential()

# Convolution layer with 32 filters of size 4x4
# input_shape is shape of each image
model.add(Conv2D(filters=32, kernel_size=(4,4),  input_shape=(28, 28, 1), activation='relu'))

# Pooling is used to reduce image dimentions
model.add(MaxPool2D(pool_size=(2, 2)))

# Flatten creates a 1D layers
model.add(Flatten())

model.add(Dense(128, activation='relu'))

# Being a muti-class classification, softmax activation function should be used
model.add(Dense(10, activation='softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

```
model.summary() # Check model summary
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 25, 25, 32)        544
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 12, 12, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 4608)              0
_________________________________________________________________
dense (Dense)                (None, 128)               589952
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
=================================================================
Total params: 591,786
Trainable params: 591,786
Non-trainable params: 0
_________________________________________________________________
```

```
# If you want to learn about early stopping check the ANN pdf.
early_stop = EarlyStopping(monitor='val_loss',patience=2)
```

```
model.fit(x_train,y_cat_train,epochs=10,validation_data=(x_test,y_cat_test),callbacks=[early_
```

```
Epoch 1/10
1875/1875 [==============================] - 32s 17ms/step - loss: 0.1417 - accuracy: 0
```

```
Epoch 2/10
1875/1875 [==============================] - 37s 19ms/step - loss: 0.0492 - accuracy: 0
Epoch 3/10
1875/1875 [==============================] - 33s 17ms/step - loss: 0.0306 - accuracy: 0
Epoch 4/10
1875/1875 [==============================] - 34s 18ms/step - loss: 0.0203 - accuracy: 0
<tensorflow.python.keras.callbacks.History at 0x7f73cc1566a0>
```

Checking & Plotting Accuracy

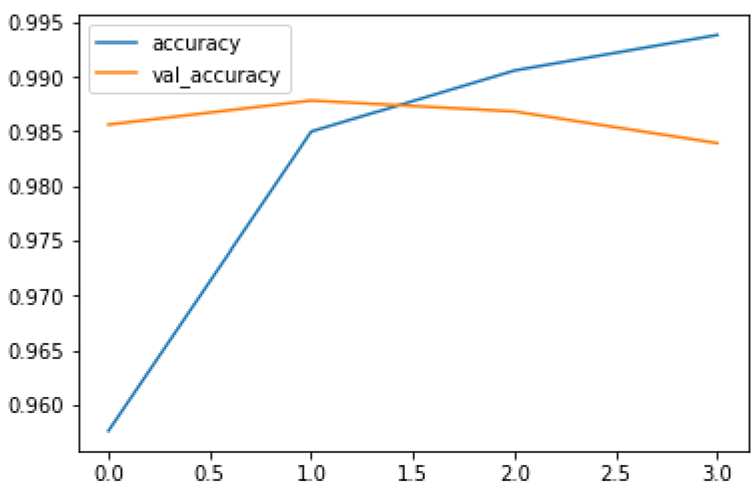
```
losses = pd.DataFrame(model.history.history)
```

```
losses.head()
```

	loss	accuracy	val_loss	val_accuracy
0	0.141731	0.957617	0.046058	0.9856
1	0.049236	0.984967	0.039530	0.9878
2	0.030635	0.990550	0.041140	0.9868
3	0.020314	0.993800	0.048296	0.9839

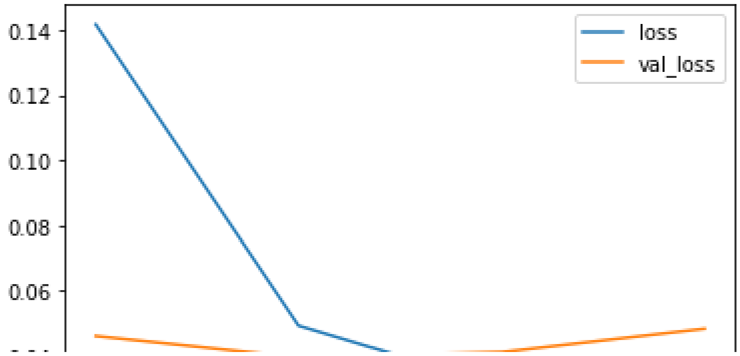
```
losses[['accuracy','val_accuracy']].plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f73c89aa550>
```




```
losses[['loss','val_loss']].plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f73c894db70>
```



Evaluating Model

```
model.evaluate(x_test,y_cat_test,verbose=0) # loss, accuracy
```

```
[0.04829579219222069, 0.9839000105857849]
```

Classification Report, Confusion Matrix

```
from sklearn.metrics import classification_report, confusion_matrix
```

```
predictions = np.argmax(model.predict(x_test), axis=-1)
```

```
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       980
           1       0.99      1.00      1.00      1135
           2       0.99      0.99      0.99      1032
           3       0.99      0.98      0.98      1010
           4       0.98      1.00      0.99       982
           5       0.94      1.00      0.97       892
           6       0.99      0.98      0.99       958
           7       0.99      0.98      0.99      1028
           8       0.97      0.99      0.98       974
           9       1.00      0.94      0.97      1009

    accuracy                           0.98     10000
   macro avg       0.98      0.98      0.98     10000
weighted avg       0.98      0.98      0.98     10000
```

```
confusion_matrix(y_test,predictions)
```

```
array([[ 974,    0,    1,    1,    0,    0,    3,    0,    1,    0],
```

```
[   0, 1132,    2,    0,    0,    0,    0,    0,    1,    0],
[   1,    1, 1023,    0,    1,    0,    0,    3,    3,    0],
[   0,    0,    1,  985,    0,   21,    0,    0,    3,    0],
[   0,    0,    0,    0,  979,    0,    1,    0,    0,    2],
[   1,    0,    0,    2,    0,  888,    0,    0,    1,    0],
[   2,    2,    0,    0,    1,   16,  935,    0,    2,    0],
[   1,    1,    9,    1,    1,    0,    0, 1009,    4,    2],
[   4,    2,    1,    2,    0,    1,    1,    0,  963,    0],
[   1,    2,    0,    1,   16,   17,    0,    4,   17,  951]])
```