CNN Report

What is CNN ?

- A convolutional neural network (CNN or convnet) is a subset of machine learning. It is one of the various types of artificial neural networks which are used for different applications and data types. A CNN is a kind of network architecture for deep learning algorithms and is specifically used for image recognition and tasks that involve the processing of pixel data.

- There are other types of neural networks in deep learning, but for identifying and recognizing objects, CNNs are the network architecture of choice. This makes them highly suitable for computer vision (CV) tasks and for applications where object recognition is vital, such as self-driving cars and facial recognition.

- The CNN is another type of neural network that can uncover key information in both time series and image data. For this reason, it is highly valuable for image-related tasks, such as image recognition, object classification and pattern recognition. To identify patterns within an image, a CNN leverages principles from linear algebra, such as matrix multiplication. CNNs can also classify audio and signal data.

CNN Layers:

- A deep learning CNN consists of three layers: a convolutional layer, a pooling layer and a fully connected (FC) layer. The convolutional layer is the first layer while the FC layer is the last.

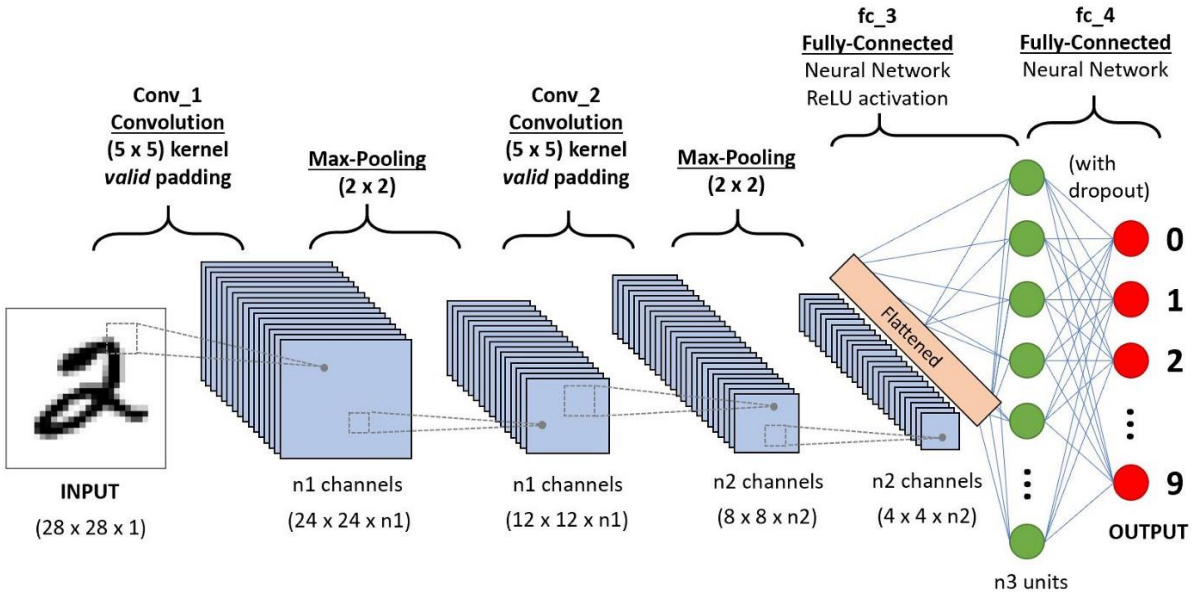
- From the convolutional layer to the FC layer, the complexity of the CNN increases. It is this increasing complexity that allows the CNN to successively identify larger portions and more complex features of an image until it finally identifies the object in its entirety.

- **Convolutional layer:** Most computations happen in the convolutional layer, which is the core building block of a CNN. A second convolutional layer can follow the initial convolutional layer. The process of convolution involves a kernel or filter inside this layer moving across the receptive fields of the image, checking if a feature is present in the image.

- **Pooling layer:** Like the convolutional layer, the pooling layer also sweeps a kernel or filter across the input image. But unlike the convolutional layer, the pooling layer reduces the number of parameters in the input and also results in some information loss. On the positive side, this layer reduces complexity and improves the efficiency of the CNN.

- **Fully connected layer.** The FC layer is where image classification happens in the CNN based on the features extracted in the previous layers. Here, *fully connected* means that all the inputs or nodes from one layer are connected to every activation unit or node of the next layer.

CNN Layers:


fc_3
Fully-Connected
Neural Network
ReLU activation
fc_4
Fully-Connected
Neural Network
Conv_1
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
Conv_2
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
(with
dropout)
Flattened
0
1
2
9
INPUT
(28 x 28 x 1)
n1 channels
(24 x 24 x n1)
n1 channels
(12 x 12 x n1)
n2 channels
(8 x 8 x n2)
n2 channels
(4 x 4 x n2)
n3 units
OUTPUT

How convolutional neural networks work ?

- A CNN can have multiple layers, each of which *learns* to detect the different features of an input image. A filter or kernel is applied to each image to produce an output that gets progressively better and more detailed after each layer. In the lower layers, the filters can start as simple features.

- At each successive layer, the filters increase in complexity to check and identify features that uniquely represent the input object. Thus, the output of each convolved image -- the partially recognized image after each layer -- becomes the input for the next layer. In the last layer, which is an FC layer, the CNN recognizes the image or the object it represents.

- With convolution, the input image goes through a set of these filters. As each filter activates certain features from the image, it does its work and passes on its output to the filter in the next layer. Each layer learns to identify different features and the operations end up being repeated for dozens, hundreds or even thousands of layers. Finally, all the image data progressing through the CNN's multiple layers allow the CNN to identify the entire object.

Code:

- **Hot encoding**

```
def create_label(image_name):
    """ Create an one-hot encoded vector from image name """
    word_label = image_name.split('.')[-2]
    #print(word_label)

    if word_label[0:1] == 'B':
        return np.array([1,0,0,0,0,0])
    elif word_label[0:1] == 'F':
        return np.array([0,1,0,0,0,0])
    elif word_label[0:1] == 'R':
        return np.array([0,0,1,0,0,0])
    elif word_label[0:1] == 'S':
        return np.array([0,0,0,1,0,0])
    elif word_label[0:1] == 'T':
        return np.array([0,0,0,0,1,0])
    elif word_label[0:1]  == 'Y':
        return np.array([0,0,0,0,0,1])
```

- **Split**

```
X_train,X_v_test, Y_train, Y_v_test = train_test_split(X, Y,
test_size=0.32)

X_test = np.array([i[0] for i in test]).reshape(-1, IMG_SIZE, IMG_SIZE,
3)
y_test = np.array([i[1] for i in test])
```

- **Resize and Reshape**

```
img_data = cv2.imread(path, cv2.IMREAD_COLOR)

img_data = cv2.cvtColor(img_data, cv2.COLOR_BGR2RGB)
img_data = cv2.resize(img_data, (IMG_SIZE, IMG_SIZE))
test_img = cv2.imread(path, cv2.IMREAD_COLOR) # [0,1,2] = [b,g,r]
test_img = test_img[:, :, [2, 1, 0]]
test_img = cv2.resize(test_img, (IMG_SIZE, IMG_SIZE))
test_img = test_img.reshape(IMG_SIZE, IMG_SIZE, 3)
X = np.array([i[0] for i in train]).reshape(-1, IMG_SIZE, IMG_SIZE, 3)
Y = np.array([i[1] for i in train])
X_test = np.array([i[0] for i in test]).reshape(-1, IMG_SIZE, IMG_SIZE, 3)
y_test = np.array([i[1] for i in test])
```

CNN Model layers :

```
tf.compat.v1.reset_default_graph()
conv_input = input_data(shape=[None, IMG_SIZE, IMG_SIZE, 3], name='input')

conv1 = conv_2d(conv_input, 32, 5, activation='relu')
pool1 = max_pool_2d(conv1, 5)

conv2 = conv_2d(pool1, 64, 5, activation='relu')
pool2 = max_pool_2d(conv2, 5)

conv3 = conv_2d(pool2, 128, 3, activation='relu')
pool3 = max_pool_2d(conv3, 5)

conv4 = conv_2d(pool3, 64, 5, activation='relu')
pool4 = max_pool_2d(conv4, 5)

conv5 = conv_2d(pool4, 32, 3, activation='relu')
pool5 = max_pool_2d(conv5, 5)

conv6 = conv_2d(pool5, 512, 5, activation='relu')
pool6 = max_pool_2d(conv6, 5)

fully_layer = fully_connected(pool6, 1024, activation='relu')
fully_layer = dropout(fully_layer, 0.5)

cnn_layers = fully_connected(fully_layer, 6, activation='softmax')

cnn_layers = regression(cnn_layers, optimizer='adam', learning_rate=LR,
loss='categorical_crossentropy', name='targets')
model = tflearn.DNN(cnn_layers, tensorboard_dir='log', tensorboard_verbose=3)
print (X_train.shape)

if (os.path.exists('model.tfl.meta')):
    model.load('./model.tfl')
else:
    model.fit({'input': X_train}, {'targets': Y_train}, batch_size=16,
n_epoch=20,
          validation_set=({'input': X_v_test}, {'targets': Y_v_test}),
          snapshot_step=440, show_metric=True, run_id=MODEL_NAME)
    model.save('model.tfl')

validate=model.evaluate(X_v_test,Y_v_test)
print('evaluate')
print(validate)
```

- **Validation**

```
sif (os.path.exists('model.tfl.meta')):
    model.load('./model.tfl')
else:
    model.fit({'input': X_train}, {'targets': Y_train}, batch_size=16,
n_epoch=20,
          validation_set=({'input': X_v_test}, {'targets': Y_v_test}),
          snapshot_step=440, show_metric=True, run_id=MODEL_NAME)
model.save('model.tfl')

validate=model.evaluate(X_v_test,Y_v_test)
print('evaluate')
print(validate)
```

- **Shuffle data**

```
    shuffle(training_data)
    np.save('train_data.npy', training_data)
    shuffle(testing_data)
    np.save('test_data.npy', testing_data)
```

- **Image size and Learning rate:**

```
IMG_SIZE = 200
LR = 0.001
```