Topic : Neural Network for Image Classification

Objective for this template:

1. To introduce participants to the basic pipeline for Image classification using a Convolutional Neural Network.
2. Demonstrate the process of training the model and evaluating its performance
3. Allow students to experiment on deriving the best performance from the prediction model by adjusting various hyperparameters.

Designed By: *Rodolfo C. Raga Jr.*

Step 1: Import needed libraries

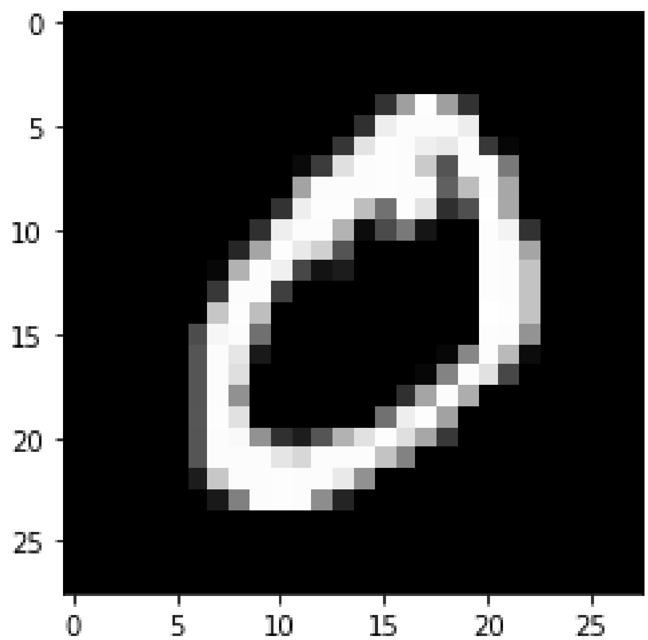
```
#KERAS
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation, Flatten
from keras.layers.convolutional import Convolution2D, MaxPooling2D
from tensorflow.keras.optimizers import SGD,RMSprop,Adam
from keras.utils import np_utils
from tensorflow.keras.utils import to_categorical

import numpy as np
import matplotlib.pyplot as plt
import matplotlib
import os
from PIL import Image
from numpy import *
# SKLEARN
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,confusion_matrix
from tensorflow.keras.datasets import mnist
from sklearn.model_selection import KFold
```

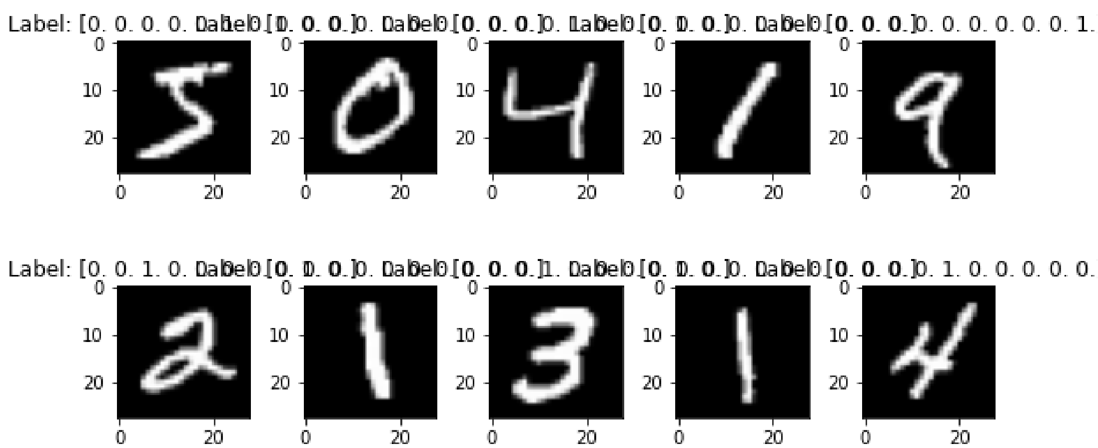
Step 2: Load and prepare dataset

```
# load dataset
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()
# reshape dataset to have a single channel
X_train = X_train.reshape((X_train.shape[0], 28, 28, 1))
X_test = X_test.reshape((X_test.shape[0], 28, 28, 1))
# one hot encode target values
```

```
Y_train = to_categorical(Y_train)
Y_test = to_categorical(Y_test)

# summarize loaded dataset
print('Train: X=%s, y=%s' % (X_train.shape, Y_train.shape))
print('Test: X=%s, y=%s' % (X_test.shape, Y_test.shape))

# pick a sample to plot
sample = 1
image = X_train[sample]# plot the sample
fig = plt.figure
plt.imshow(np.squeeze(image), cmap='gray')
plt.show()

num_row = 2
num_col = 5# plot images
num=10
images = X_train[:num]
labels = Y_train[:num]
fig, axes = plt.subplots(num_row, num_col,
figsize=(1.5*num_col,2*num_row))
for i in range(num):
    ax = axes[i//num_col, i%num_col]
    ax.imshow(np.squeeze(images[i]), cmap='gray')
    ax.set_title('Label: {}'.format(labels[i]))
plt.tight_layout()
plt.show()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-
keras-datasets/mnist.npz
11493376/11490434 [==============================] - 7s 1us/step
11501568/11490434 [==============================] - 7s 1us/step
Train: X=(60000, 28, 28, 1), y=(60000, 10)
Test: X=(10000, 28, 28, 1), y=(10000, 10)
```





Step 3 : Normalize Pixel Data to Range (0-1)

```
# convert from integers to floats
train_norm = X_train.astype('float32')
test_norm = X_test.astype('float32')
# normalize to range 0-1
train_norm = train_norm / 255.0
test_norm = test_norm / 255.0
```

Step 4: Define Convolutional Neural Network model

```
# number of convolutional filters to use
nb_filters = 32
# size of pooling area for max pooling
nb_pool = 2
# convolution kernel size
```

```
nb_conv = 3

#batch_size to train
batch_size = 32
# number of output classes
nb_classes = 3
# number of epochs to train
nb_epoch = 5

model = Sequential()
model = Sequential()
model.add(Convolution2D(32, (3, 3), activation='relu',
kernel_initializer='he_uniform', input_shape=(28, 28, 1)))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(100, activation='relu',
kernel_initializer='he_uniform'))
model.add(Dense(10, activation='softmax'))


model.compile(optimizer=keras.optimizers.Adadelta(learning_rate=0.0001
),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Step 5: Train the model and gather training history data

```
hist = model.fit(X_train, Y_train, batch_size=batch_size,
epochs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

Epoch 1/5
1875/1875 [==============================] - 41s 21ms/step - loss:
208.6384 - accuracy: 0.0908 - val_loss: 166.5787 - val_accuracy:
0.1050
Epoch 2/5
1875/1875 [==============================] - 40s 21ms/step - loss:
137.7864 - accuracy: 0.1302 - val_loss: 114.7851 - val_accuracy:
0.1545
Epoch 3/5
1875/1875 [==============================] - 40s 21ms/step - loss:
97.3053 - accuracy: 0.1931 - val_loss: 82.4091 - val_accuracy: 0.2270
Epoch 4/5
1875/1875 [==============================] - 40s 21ms/step - loss:
71.6283 - accuracy: 0.2734 - val_loss: 61.7708 - val_accuracy: 0.3160
Epoch 5/5
1875/1875 [==============================] - 41s 22ms/step - loss:
55.1828 - accuracy: 0.3544 - val_loss: 48.6382 - val_accuracy: 0.3959
```

Step 6: Display results of performance assessment metrics

```
score = model.evaluate(X_test, Y_test, verbose=0)
print(model.metrics_names)
print(score)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

print(Y_test)
Y_pred = model.predict(X_test)
print(Y_pred)
y_pred = np.argmax(Y_pred, axis=1)
print(y_pred)
target_names = ['class 0(Comfort)', 'class 1(Discomfort)', 'class
2(HORSES)']
print("Performance report: \
n",classification_report(np.argmax(Y_test,axis=1), y_pred,labels=[0,1,
2, 3,4,5,6,7,8,9]))
print("Confusion Matrix: \
n",confusion_matrix(np.argmax(Y_test,axis=1), y_pred))
```

```
['loss', 'accuracy']
[48.638153076171875, 0.39590001106262207]
Test loss: 48.638153076171875
Test accuracy: 0.39590001106262207
[[0. 0. 0. ... 1. 0. 0.]
 [0. 0. 1. ... 0. 0. 0.]
 [0. 1. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
[[0.00000000e+00 1.22936867e-38 1.41566492e-22 ... 1.00000000e+00
  0.00000000e+00 9.98810002e-17]
 [2.35039159e-25 1.34993905e-38 4.08892147e-33 ... 0.00000000e+00
  0.00000000e+00 4.26124700e-07]
 [5.43401841e-37 2.56945984e-03 2.74293475e-21 ... 4.56308209e-33
  3.61202930e-37 2.79900259e-18]
 ...
 [0.00000000e+00 1.03753474e-20 0.00000000e+00 ... 0.00000000e+00
  0.00000000e+00 2.83618727e-24]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 ... 0.00000000e+00
  0.00000000e+00 1.00000000e+00]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 ... 0.00000000e+00
  0.00000000e+00 0.00000000e+00]]
[7 6 6 ... 4 9 6]
Performance report:
               precision    recall  f1-score   support

           0       0.67      0.66      0.67       980
           1       0.59      0.56      0.58      1135
```

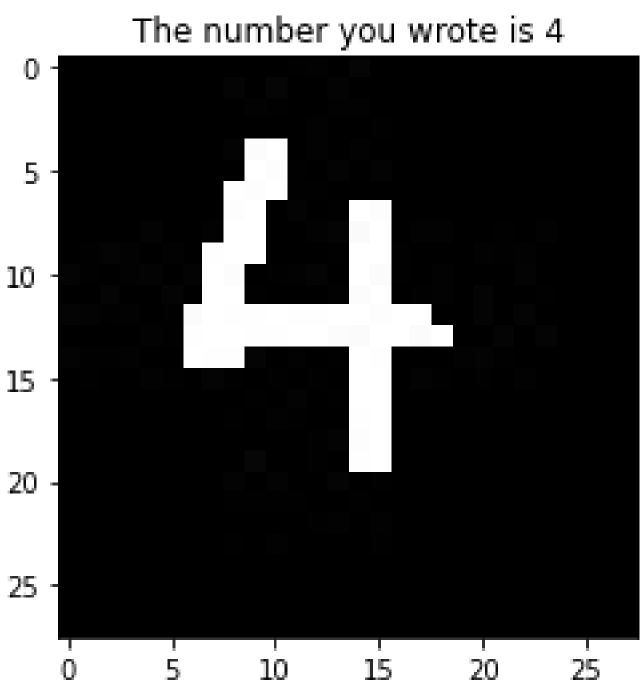
```
           2       0.37      0.42      0.39      1032
           3       0.23      0.27      0.25      1010
           4       0.32      0.30      0.31       982
           5       0.29      0.25      0.27       892
           6       0.45      0.52      0.48       958
           7       0.52      0.30      0.38      1028
           8       0.21      0.22      0.21       974
           9       0.36      0.44      0.39      1009

    accuracy                           0.40     10000
   macro avg       0.40      0.39      0.39     10000
weighted avg       0.40      0.40      0.40     10000

Confusion Matrix:
 [[647   1  31  88   2 114  41   1  25  30]
 [  2 639  33  46   5   2 103 104 127  74]
 [123  69 435 148  18  37  58   5  63  76]
 [ 22  31 126 269 116  61 178  56 116  35]
 [  8  11 107  53 292  89  60  28 152 182]
 [ 44  67  52 151  53 221  82  24  69 129]
 [ 33   4  34  61  59 165 496  10  67  29]
 [ 47 124 134  21 176   9  17 308  54 138]
 [ 16 115 160 254  24  55  29   6 212 103]
 [ 20  19  70  65 162  17  43  52 121 440]]
```

Step 6: Design and Test Deployment Interface

```
from skimage import color
from skimage import io
from IPython.display import clear_output
while True:
    img=color.rgb2gray(io.imread("testnum.jpg"))
    print("Image Shape")
    print(img.shape)
    print(type(img))
    print("size"+str(img.size))
    plt.imshow(img, cmap='gray')
    img = np.array(img)
    img.reshape([-1, 28, 28, 1])
    print("reshaped")
    print("img shape",img.shape)
    maxidx = model.predict(img.reshape([-1, 28, 28, 1]))
    print("maxidx shape",maxidx.shape)
    print(maxidx)
    print(maxidx.argmax())
    max_ind = np.array(maxidx)
    print(max_ind.shape)
    max_val = max_ind[0,maxidx.argmax()]
    print(max_val)
    i, j = np.unravel_index(maxidx.argmax(),maxidx.shape)
```

```
    plt.title('The number you wrote is '+repr(j))
    plt.show()
    s_continue = input("Try again (Y/N)?: ")
    if (s_continue=='N'):
        break
    clear_output(wait=True)
```

```
Image Shape
(28, 28)
<class 'numpy.ndarray'>
size784
reshaped
img shape (28, 28)
maxidx shape (1, 10)
[[0.08995578 0.05939199 0.12595616 0.10388833 0.1642152  0.07654279
  0.07967021 0.10153501 0.07829402 0.12055047]]
4
(1, 10)
0.1642152
```




```
Try again (Y/N)?: N
```

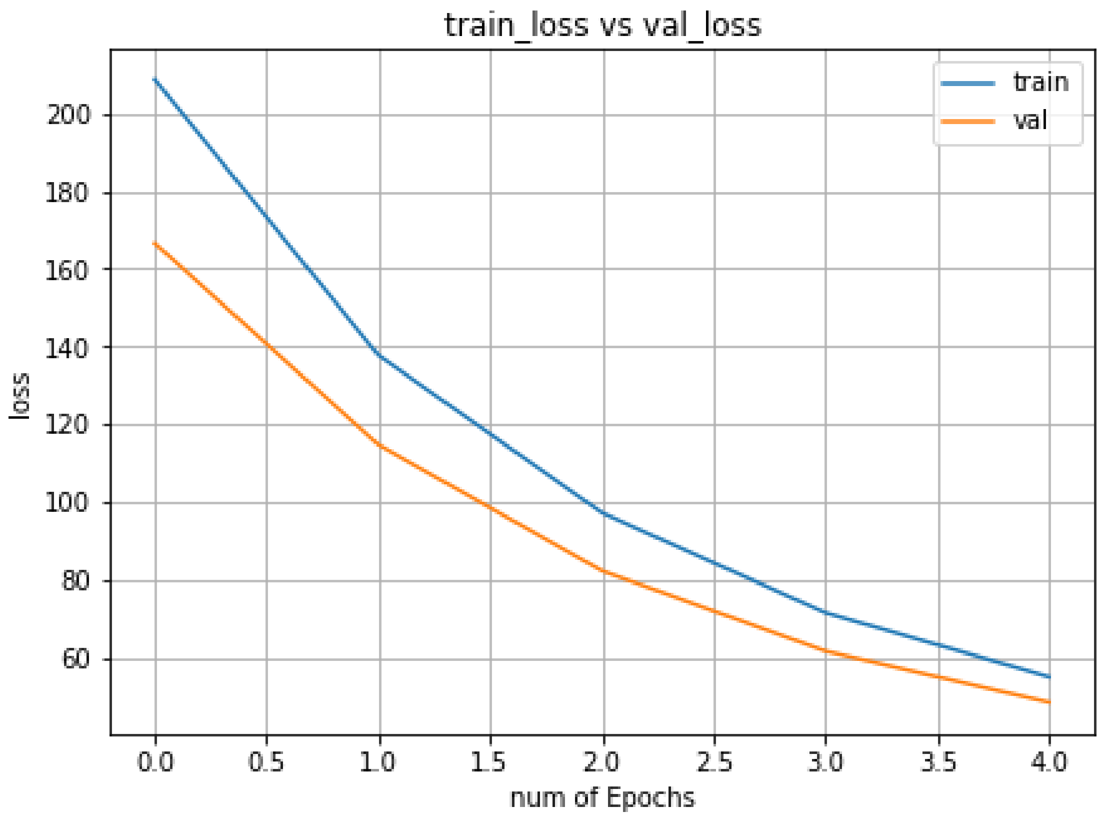
Other things we can do:

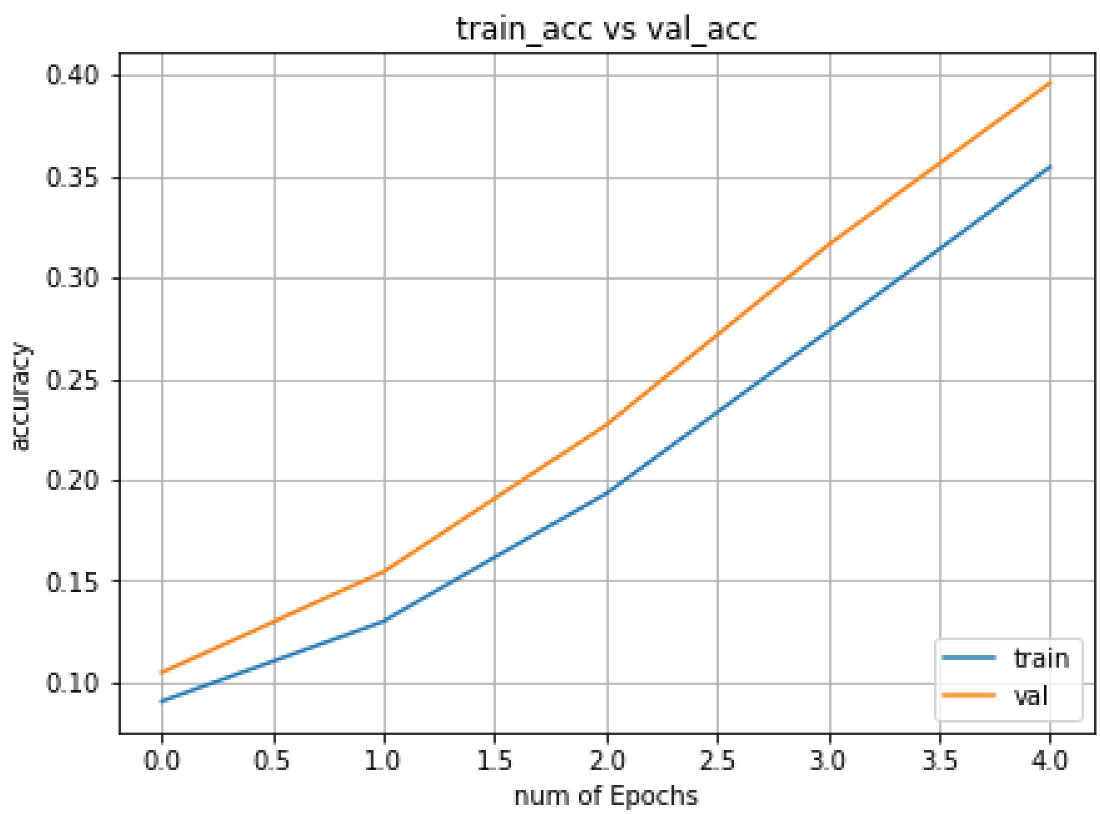
Step 7: Analyze training statistics

```
train_loss=hist.history['loss']
val_loss=hist.history['val_loss']
train_acc=hist.history['accuracy']
val_acc=hist.history['val_accuracy']
```

```python
xc=range(nb_epoch)

plt.figure(1,figsize=(7,5))
plt.plot(xc,train_loss)
plt.plot(xc,val_loss)
plt.xlabel('num of Epochs')
plt.ylabel('loss')
plt.title('train_loss vs val_loss')
plt.grid(True)
plt.legend(['train','val'])
#print (plt.style.available # use bmh, classic,ggplot for big
pictures)
#plt.style.use(['classic'])

plt.figure(2,figsize=(7,5))
plt.plot(xc,train_acc)
plt.plot(xc,val_acc)
plt.xlabel('num of Epochs')
plt.ylabel('accuracy')
plt.title('train_acc vs val_acc')
plt.grid(True)
plt.legend(['train','val'],loc=4)
#print plt.style.available # use bmh, classic,ggplot for big pictures
#plt.style.use(['classic'])
```

```
<matplotlib.legend.Legend at 0x203c59841f0>
```


train_loss vs val_loss
train
val
loss
num of Epochs


train_acc vs val_acc
train
val
accuracy
num of Epochs