Doc1_Assignment3_Q1_Q4_dec657

November 21, 2020

Importing Required Libraries

[1]:
```
# Packages
import datetime
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
# Tensorflow packages
from tensorflow.keras import Model
from tensorflow.keras.models import Sequential
from tensorflow.keras.losses import categorical_crossentropy
from tensorflow.keras.layers import Dense, Flatten, Conv2D, AveragePooling2D
from tensorflow.keras.regularizers import l2, l1
from tensorflow.keras.layers import GlobalMaxPooling2D
from tensorflow.keras import datasets
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import Callback
from tensorflow.keras.models import load_model
import math
```

Data Loading and Spliting

[2]:
```
(x_train, y_train), (x_test, y_test) = datasets.fashion_mnist.load_data()
```

Data Preparation

[3]:
```
# Fixation of Axis for the dataset
x_train = x_train[:, :, :, np.newaxis]
x_test = x_test[:, :, :, np.newaxis]

# Binary classes
num_classes = 10
y_train = to_categorical(y_train, num_classes)
y_test = to_categorical(y_test, num_classes)

# Normalization
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
```

```
x_test /= 255
```

Create Model

[4]:

```
# LeNet5 base model layers
class LeNet(Sequential):
    def __init__(self, input_shape, nb_classes):
        super().__init__()
        self.add(Conv2D(6, kernel_size=(5, 5), strides=(1, 1),␣
 ↪activation='relu', input_shape=input_shape, padding="same"))
        self.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2),␣
 ↪padding='valid'))
        self.add(Conv2D(16, kernel_size=(5, 5), strides=(1, 1),␣
 ↪activation='relu', padding='valid'))
        self.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2),␣
 ↪padding='valid'))
        self.add(Flatten())
        self.add(Dense(120, activation='relu'))
        self.add(Dense(84, activation='relu'))
        self.add(Dense(nb_classes, activation='softmax'))
        self.compile(optimizer='adam',
                     loss=categorical_crossentropy,
                     metrics=['accuracy'])
```

[5]:

```
# LeNet5 model layers with L2 weight decay regularization
class LeNetReguL2(Sequential):
    def __init__(self, input_shape, nb_classes, l2_value = 0.01):
        super().__init__()
        self.add(Conv2D(6, kernel_size=(5, 5), strides=(1, 1),␣
 ↪activation='relu', input_shape=input_shape, padding="same",␣
 ↪kernel_regularizer=l2(l2_value)))
        self.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2),␣
 ↪padding='valid'))
        self.add(Conv2D(16, kernel_size=(5, 5), strides=(1, 1),␣
 ↪activation='relu', padding='valid', kernel_regularizer=l2(l2_value)))
        self.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2),␣
 ↪padding='valid'))
        self.add(Flatten())
        self.add(Dense(120, activation='relu', kernel_regularizer=l2(l2_value)))
        self.add(Dense(84, activation='relu', kernel_regularizer=l2(l2_value)))
        self.add(Dense(nb_classes, activation='softmax'))

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

[6]:

```
# LeNet5 model layers with L1 weight decay regularization
class LeNetReguL1(Sequential):
    def __init__(self, input_shape, nb_classes, l1_value = 0.01):
        super().__init__()
        self.add(Conv2D(6, kernel_size=(5, 5), strides=(1, 1),␣
 ↪activation='relu', input_shape=input_shape, padding="same",␣
 ↪kernel_regularizer=l1(l1_value)))
        self.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2),␣
 ↪padding='valid'))
        self.add(Conv2D(16, kernel_size=(5, 5), strides=(1, 1),␣
 ↪activation='relu', padding='valid', kernel_regularizer=l1(l1_value)))
        self.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2),␣
 ↪padding='valid'))
        self.add(Flatten())
        self.add(Dense(120, activation='relu', kernel_regularizer=l1(l1_value)))
        self.add(Dense(84, activation='relu', kernel_regularizer=l1(l1_value)))
        self.add(Dense(nb_classes, activation='softmax'))

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

[7]:

```
# LeNet5 model without fully-connected layers
class LeNetGAP(Sequential):
    def __init__(self, input_shape, nb_classes):
        super().__init__()
        self.add(Conv2D(6, kernel_size=(5, 5), strides=(1, 1),␣
 ↪activation='relu', input_shape=input_shape, padding="same"))
        self.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2),␣
 ↪padding='valid'))
        self.add(Conv2D(16, kernel_size=(5, 5), strides=(1, 1),␣
 ↪activation='relu', padding='valid'))
        self.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2),␣
 ↪padding='valid'))
        self.add(GlobalMaxPooling2D())
        # self.add(GlobalMaxPooling2D())
        self.add(Dense(nb_classes, activation='softmax'))
        self.compile(optimizer='adam',
                    loss=categorical_crossentropy,
                    metrics=['accuracy'])
```

[8]:

```
# # LeNet5 model experiments layers
# class Temp(Sequential):
#     def __init__(self, input_shape, nb_classes):
#         super().__init__()
#         # self.add(Conv2D(6, kernel_size=(5, 5), strides=(1, 1),␣
↪activation='relu', input_shape=input_shape, padding="same",␣
↪kernel_regularizer=l2(0.01)))
#         self.add(Conv2D(6, kernel_size=(5, 5), strides=(1, 1),␣
↪activation='relu', input_shape=input_shape, padding="same",␣
↪kernel_regularizer=l1(0.01)))
#         self.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2),␣
↪padding='valid'))
#         # self.add(Conv2D(16, kernel_size=(5, 5), strides=(1, 1),␣
↪activation='relu', padding='valid', kernel_regularizer=l2(0.01)))
#         self.add(Conv2D(16, kernel_size=(5, 5), strides=(1, 1),␣
↪activation='relu', padding='valid', kernel_regularizer=l1(0.01)))
#         self.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2),␣
↪padding='valid'))
#         # self.add(Flatten())
#         # self.add(Dense(120, activation='relu'))
#         self.add(GlobalMaxPooling2D())
#         # self.add(Dense(84, activation='relu'))
#         # self.add(GlobalMaxPooling2D())
#         self.add(Dense(nb_classes, activation='softmax'))
#         # self.add(AveragePooling2D())
#         self.compile(optimizer='adam',
#                      loss=categorical_crossentropy,
#                      metrics=['accuracy'])
```

Sparsity Coefficient

[9]:

```
# def gini_coefficient(x):
#     mad = np.abs(np.subtract.outer(x, x)).mean()
#     rmad = mad/np.mean(x)
#     g = 0.5 * rmad
#     return g

def hoyer_index(x):
  rows,columns = x.shape
  f_up = 0
  f_down = 0
  for row in range (rows):
    for column in range (columns):
      if x[row, column] != 0:
        f_up = f_up + np.absolute(x[row, column])
        f_down = f_down + (x[row, column])**2
  f_down = (f_down)**(-1/2)
  h_index = ((rows*columns)**(-1/2) - (f_up/f_down))/((rows*columns)**(-1/2)-1)
  return h_index
```

```
# x = np.zeros((10,10))
# x [2,2] = 1
# hoyer_index(x)
```

Construct Model

[10]:

```
modelLeNet = LeNet(x_train[0].shape, num_classes)
modelLeNetReguL2v1 = LeNetReguL2(x_train[0].shape, num_classes, l2_value = 0.01)
modelLeNetReguL2v2 = LeNetReguL2(x_train[0].shape, num_classes, l2_value = 0.02)
modelLeNetReguL1v1 = LeNetReguL1(x_train[0].shape, num_classes, l1_value = 0.01)
modelLeNetReguL1v2 = LeNetReguL1(x_train[0].shape, num_classes, l1_value = 0.01)
modelLeNetGAP = LeNetGAP(x_train[0].shape, num_classes)
# model.summary()
# !pip install ipdb
# import ipdb; ipdb.set_trace()
```

Timing

[11]:

```
import time
class TimeHistory(tf.keras.callbacks.Callback):
    test_time_start = 0
    def on_train_begin(self, logs={}):
        self.times = []

    def on_test_begin(self, batch, logs={}):
        self.test_time_start = time.time()

    def on_test_end(self, batch, logs={}):
        self.times.append(time.time() - self.test_time_start)
# reference: https://stackoverflow.com/questions/43178668/
 ↪record-the-computation-time-for-each-epoch-in-keras-during-model-fit
```

Training the model

[12]:

```
run_times_LeNet = []
run_times_LeNetReguL2v1 = []
run_times_LeNetReguL2v2 = []
run_times_LeNetReguL1v1 = []
run_times_LeNetReguL1v2 = []
run_times_LeNetGAP = []
no_of_epochs = 5

for i in range(0, 3):
  time_callback = TimeHistory()
  log_dir="logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
  tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,␣
 ↪histogram_freq=1)
  historyLeNet = modelLeNet.fit(x_train, y=y_train,
          epochs=no_of_epochs,
          validation_data=(x_test, y_test),
          callbacks=[tensorboard_callback, time_callback],
          )
run_times_LeNet.append(historyLeNet.history['val_loss'][-1])
model_info_LeNet = {'TrainingError':historyLeNet.
↪history['loss'][-1],'TestError': historyLeNet.history['val_loss'][-1],␣
↪'SDTestError':run_times_LeNet, 'InferenceTime':time_callback.times[-1],␣
↪'NoofParameters':modelLeNet.count_params()}
modelLeNet.save('modelLeNet.h5')
# del modelLeNet

time_callback = TimeHistory()
log_dir="logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,␣
↪histogram_freq=1)
historyLeNetReguL2v1 = modelLeNetReguL2v1.fit(x_train, y=y_train,
          epochs=no_of_epochs,
          validation_data=(x_test, y_test),
          callbacks=[tensorboard_callback, time_callback],
          )
run_times_LeNetReguL2v1.append(historyLeNetReguL2v1.history['val_loss'][-1])
model_info_LeNetReguL2v1 = {'TrainingError':historyLeNetReguL2v1.
↪history['loss'][-1],'TestError': historyLeNetReguL2v1.
↪history['val_loss'][-1], 'SDTestError':run_times_LeNetReguL2v1,␣
↪'InferenceTime':time_callback.times[-1], 'NoofParameters':modelLeNetReguL2v1.
↪count_params()}
modelLeNetReguL2v1.save('modelLeNetReguL2v1.h5')
# del modelLeNetReguL2v1


time_callback = TimeHistory()
log_dir="logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,␣
↪histogram_freq=1)
historyLeNetReguL2v2 = modelLeNetReguL2v2.fit(x_train, y=y_train,
          epochs=no_of_epochs,
          validation_data=(x_test, y_test),
          callbacks=[tensorboard_callback, time_callback],
          )
run_times_LeNetReguL2v2.append(historyLeNetReguL2v2.history['val_loss'][-1])
model_info_LeNetReguL2v2 = {'TrainingError':historyLeNetReguL2v2.
↪history['loss'][-1],'TestError': historyLeNetReguL2v2.
↪history['val_loss'][-1], 'SDTestError':run_times_LeNetReguL2v2,␣
↪'InferenceTime':time_callback.times[-1], 'NoofParameters':modelLeNetReguL2v2.
↪count_params()}
modelLeNetReguL2v2.save('modelLeNetReguL2v2.h5')
```

```
 # del modelLeNetReguL2v2

 time_callback = TimeHistory()
 log_dir="logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
 tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,␣
↪histogram_freq=1)
 historyLeNetReguL1v1 = modelLeNetReguL1v1.fit(x_train, y=y_train,
           epochs=no_of_epochs,
           validation_data=(x_test, y_test),
           callbacks=[tensorboard_callback, time_callback],
           )
 run_times_LeNetReguL1v1.append(historyLeNetReguL1v1.history['val_loss'][-1])
 model_info_LeNetReguL1v1 = {'TrainingError':historyLeNetReguL1v1.
↪history['loss'][-1],'TestError': historyLeNetReguL1v1.
↪history['val_loss'][-1], 'SDTestError':run_times_LeNetReguL1v1,␣
↪'InferenceTime':time_callback.times[-1], 'NoofParameters':modelLeNetReguL1v1.
↪count_params()}
 modelLeNetReguL1v1.save('modelLeNetReguL1v1.h5')
 # del modelLeNetReguL1v1


 time_callback = TimeHistory()
 log_dir="logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
 tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,␣
↪histogram_freq=1)
 historyLeNetReguL1v2 = modelLeNetReguL1v2.fit(x_train, y=y_train,
           epochs=no_of_epochs,
           validation_data=(x_test, y_test),
           callbacks=[tensorboard_callback, time_callback],
           )
 run_times_LeNetReguL1v2.append(historyLeNetReguL1v2.history['val_loss'][-1])
 model_info_LeNetReguL1v2 = {'TrainingError':historyLeNetReguL1v2.
↪history['loss'][-1],'TestError': historyLeNetReguL1v2.
↪history['val_loss'][-1], 'SDTestError':run_times_LeNetReguL1v2,␣
↪'InferenceTime':time_callback.times[-1], 'NoofParameters':modelLeNetReguL1v2.
↪count_params()}
 modelLeNetReguL1v2.save('modelLeNetReguL1v2.h5')
 # del modelLeNetReguL1v2

 time_callback = TimeHistory()
 log_dir="logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
 tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,␣
↪histogram_freq=1)
 historyLeNetGAP = modelLeNetGAP.fit(x_train, y=y_train,
           epochs=no_of_epochs,
           validation_data=(x_test, y_test),
           callbacks=[tensorboard_callback, time_callback],
           )
 run_times_LeNetGAP.append(historyLeNetGAP.history['val_loss'][-1])
 model_info_LeNetGAP = {'TrainingError':historyLeNetGAP.
↪history['loss'][-1],'TestError': historyLeNetGAP.history['val_loss'][-1],␣
↪'SDTestError':run_times_LeNetGAP, 'InferenceTime':time_callback.times[-1],␣
↪'NoofParameters':modelLeNetGAP.count_params()}
 modelLeNetGAP.save('modelLeNetGAP.h5')
 # del modelLeNetGAP
```

```
Epoch 1/5
   1/1875 […] - ETA: 0s - loss: 2.2845 - accuracy:
0.1562WARNING:tensorflow:From /home/ubuntu/anaconda3/lib/python3.8/site-
packages/tensorflow/python/ops/summary_ops_v2.py:1277: stop (from
tensorflow.python.eager.profiler) is deprecated and will be removed after
2020-07-01.
Instructions for updating:
use `tf.profiler.experimental.stop` instead.
1875/1875 [==============================] - 12s 6ms/step - loss: 0.5478 -
accuracy: 0.7973 - val_loss: 0.4485 - val_accuracy: 0.8342
Epoch 2/5
1875/1875 [==============================] - 11s 6ms/step - loss: 0.3734 -
accuracy: 0.8614 - val_loss: 0.3646 - val_accuracy: 0.8631
Epoch 3/5
1875/1875 [==============================] - 11s 6ms/step - loss: 0.3238 -
accuracy: 0.8804 - val_loss: 0.3427 - val_accuracy: 0.8735
Epoch 4/5
1875/1875 [==============================] - 11s 6ms/step - loss: 0.2931 -
accuracy: 0.8908 - val_loss: 0.3339 - val_accuracy: 0.8777
Epoch 5/5
1875/1875 [==============================] - 11s 6ms/step - loss: 0.2707 -
accuracy: 0.8996 - val_loss: 0.2986 - val_accuracy: 0.8904
Epoch 1/5
   1/1875 […] - ETA: 0s - loss: 5.2646 - accuracy:
0.0625WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow compared
to the batch time (batch time: 0.0077s vs `on_train_batch_end` time: 0.0341s).
Check your callbacks.
1875/1875 [==============================] - 12s 6ms/step - loss: 1.0966 -
accuracy: 0.7432 - val_loss: 0.7909 - val_accuracy: 0.7945
Epoch 2/5
1875/1875 [==============================] - 11s 6ms/step - loss: 0.7566 -
accuracy: 0.7973 - val_loss: 0.7308 - val_accuracy: 0.8010
Epoch 3/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.7001 -
accuracy: 0.8122 - val_loss: 0.7057 - val_accuracy: 0.8053
Epoch 4/5
1875/1875 [==============================] - 11s 6ms/step - loss: 0.6703 -
```

```
accuracy: 0.8188 - val_loss: 0.6902 - val_accuracy: 0.8132
Epoch 5/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.6486 -
accuracy: 0.8262 - val_loss: 0.6423 - val_accuracy: 0.8201
Epoch 1/5
   1/1875 […] - ETA: 0s - loss: 8.1646 - accuracy:
0.0938WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow compared
to the batch time (batch time: 0.0057s vs `on_train_batch_end` time: 0.0323s).
Check your callbacks.
1875/1875 [==============================] - 12s 6ms/step - loss: 1.2560 -
accuracy: 0.7089 - val_loss: 0.9240 - val_accuracy: 0.7475
Epoch 2/5
1875/1875 [==============================] - 11s 6ms/step - loss: 0.8626 -
accuracy: 0.7517 - val_loss: 0.8342 - val_accuracy: 0.7633
Epoch 3/5
1875/1875 [==============================] - 11s 6ms/step - loss: 0.8122 -
accuracy: 0.7652 - val_loss: 0.8089 - val_accuracy: 0.7635
Epoch 4/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.7823 -
accuracy: 0.7721 - val_loss: 0.7934 - val_accuracy: 0.7638
Epoch 5/5
1875/1875 [==============================] - 11s 6ms/step - loss: 0.7613 -
accuracy: 0.7795 - val_loss: 0.7866 - val_accuracy: 0.7633
Epoch 1/5
   1/1875 […] - ETA: 0s - loss: 38.1404 - accuracy:
0.0938WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow compared
to the batch time (batch time: 0.0062s vs `on_train_batch_end` time: 0.0359s).
Check your callbacks.
1875/1875 [==============================] - 12s 6ms/step - loss: 2.2111 -
accuracy: 0.6416 - val_loss: 1.1806 - val_accuracy: 0.7244
Epoch 2/5
1875/1875 [==============================] - 11s 6ms/step - loss: 1.1266 -
accuracy: 0.7277 - val_loss: 1.1075 - val_accuracy: 0.7362
Epoch 3/5
1875/1875 [==============================] - 11s 6ms/step - loss: 1.0498 -
accuracy: 0.7386 - val_loss: 1.0518 - val_accuracy: 0.7380
Epoch 4/5
1875/1875 [==============================] - 11s 6ms/step - loss: 1.0188 -
accuracy: 0.7423 - val_loss: 1.0743 - val_accuracy: 0.7299
Epoch 5/5
1875/1875 [==============================] - 11s 6ms/step - loss: 0.9989 -
accuracy: 0.7480 - val_loss: 1.0137 - val_accuracy: 0.7471
Epoch 1/5
   1/1875 […] - ETA: 0s - loss: 37.9966 - accuracy:
0.2500WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow compared
to the batch time (batch time: 0.0061s vs `on_train_batch_end` time: 0.0353s).
Check your callbacks.
1875/1875 [==============================] - 12s 6ms/step - loss: 2.2619 -
```

```
accuracy: 0.6551 - val_loss: 1.2049 - val_accuracy: 0.7193
Epoch 2/5
1875/1875 [==============================] - 11s 6ms/step - loss: 1.1513 -
accuracy: 0.7224 - val_loss: 1.1299 - val_accuracy: 0.7140
Epoch 3/5
1875/1875 [==============================] - 11s 6ms/step - loss: 1.0822 -
accuracy: 0.7319 - val_loss: 1.0704 - val_accuracy: 0.7380
Epoch 4/5
1875/1875 [==============================] - 12s 6ms/step - loss: 1.0415 -
accuracy: 0.7388 - val_loss: 1.0529 - val_accuracy: 0.7264
Epoch 5/5
1875/1875 [==============================] - 11s 6ms/step - loss: 1.0199 -
accuracy: 0.7445 - val_loss: 1.0187 - val_accuracy: 0.7468
Epoch 1/5
   1/1875 […] - ETA: 0s - loss: 2.3424 - accuracy:
0.1250WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow compared
to the batch time (batch time: 0.0057s vs `on_train_batch_end` time: 0.0266s).
Check your callbacks.
1875/1875 [==============================] - 11s 6ms/step - loss: 0.8606 -
accuracy: 0.7018 - val_loss: 0.6298 - val_accuracy: 0.7788
Epoch 2/5
1875/1875 [==============================] - 11s 6ms/step - loss: 0.5750 -
accuracy: 0.7976 - val_loss: 0.6074 - val_accuracy: 0.7769
Epoch 3/5
1875/1875 [==============================] - 11s 6ms/step - loss: 0.5162 -
accuracy: 0.8181 - val_loss: 0.5199 - val_accuracy: 0.8191
Epoch 4/5
1875/1875 [==============================] - 11s 6ms/step - loss: 0.4825 -
accuracy: 0.8308 - val_loss: 0.5324 - val_accuracy: 0.8128
Epoch 5/5
1875/1875 [==============================] - 11s 6ms/step - loss: 0.4613 -
accuracy: 0.8369 - val_loss: 0.4784 - val_accuracy: 0.8328
Epoch 1/5
   1/1875 […] - ETA: 0s - loss: 0.2394 - accuracy:
0.9375WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow compared
to the batch time (batch time: 0.0066s vs `on_train_batch_end` time: 0.0121s).
Check your callbacks.
1875/1875 [==============================] - 12s 6ms/step - loss: 0.2540 -
accuracy: 0.9035 - val_loss: 0.3016 - val_accuracy: 0.8907
Epoch 2/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.2376 -
accuracy: 0.9107 - val_loss: 0.2865 - val_accuracy: 0.8966
Epoch 3/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.2246 -
accuracy: 0.9161 - val_loss: 0.2808 - val_accuracy: 0.8982
Epoch 4/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.2129 -
accuracy: 0.9198 - val_loss: 0.2831 - val_accuracy: 0.8983
```

```
Epoch 5/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.2045 -
accuracy: 0.9227 - val_loss: 0.2899 - val_accuracy: 0.8958
Epoch 1/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.6343 -
accuracy: 0.8274 - val_loss: 0.6448 - val_accuracy: 0.8213
Epoch 2/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.6191 -
accuracy: 0.8312 - val_loss: 0.6355 - val_accuracy: 0.8241
Epoch 3/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.6063 -
accuracy: 0.8352 - val_loss: 0.6201 - val_accuracy: 0.8276
Epoch 4/5
1875/1875 [==============================] - 12s 7ms/step - loss: 0.5973 -
accuracy: 0.8371 - val_loss: 0.6107 - val_accuracy: 0.8316
Epoch 5/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.5888 -
accuracy: 0.8392 - val_loss: 0.6392 - val_accuracy: 0.8126
Epoch 1/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.7438 -
accuracy: 0.7865 - val_loss: 0.7782 - val_accuracy: 0.7799
Epoch 2/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.7297 -
accuracy: 0.7906 - val_loss: 0.7521 - val_accuracy: 0.7808
Epoch 3/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.7178 -
accuracy: 0.7955 - val_loss: 0.7273 - val_accuracy: 0.7913
Epoch 4/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.7065 -
accuracy: 0.8008 - val_loss: 0.7132 - val_accuracy: 0.7957
Epoch 5/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.7001 -
accuracy: 0.8020 - val_loss: 0.7377 - val_accuracy: 0.7905
Epoch 1/5
1875/1875 [==============================] - 12s 7ms/step - loss: 0.9875 -
accuracy: 0.7470 - val_loss: 0.9965 - val_accuracy: 0.7485
Epoch 2/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.9787 -
accuracy: 0.7499 - val_loss: 0.9823 - val_accuracy: 0.7491
Epoch 3/5
1875/1875 [==============================] - 13s 7ms/step - loss: 0.9732 -
accuracy: 0.7511 - val_loss: 0.9788 - val_accuracy: 0.7450
Epoch 4/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.9662 -
accuracy: 0.7528 - val_loss: 0.9840 - val_accuracy: 0.7452
Epoch 5/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.9599 -
accuracy: 0.7538 - val_loss: 0.9699 - val_accuracy: 0.7505
```

```
Epoch 1/5
1875/1875 [==============================] - 12s 6ms/step - loss: 1.0059 -
accuracy: 0.7468 - val_loss: 1.0325 - val_accuracy: 0.7386
Epoch 2/5
1875/1875 [==============================] - 13s 7ms/step - loss: 0.9939 -
accuracy: 0.7510 - val_loss: 1.0206 - val_accuracy: 0.7415
Epoch 3/5
1875/1875 [==============================] - 13s 7ms/step - loss: 0.9873 -
accuracy: 0.7520 - val_loss: 1.0137 - val_accuracy: 0.7287
Epoch 4/5
1875/1875 [==============================] - 13s 7ms/step - loss: 0.9783 -
accuracy: 0.7536 - val_loss: 1.0087 - val_accuracy: 0.7402
Epoch 5/5
1875/1875 [==============================] - 13s 7ms/step - loss: 0.9709 -
accuracy: 0.7561 - val_loss: 0.9781 - val_accuracy: 0.7487
Epoch 1/5
1875/1875 [==============================] - 11s 6ms/step - loss: 0.4440 -
accuracy: 0.8420 - val_loss: 0.4644 - val_accuracy: 0.8350
Epoch 2/5
1875/1875 [==============================] - 11s 6ms/step - loss: 0.4317 -
accuracy: 0.8464 - val_loss: 0.4629 - val_accuracy: 0.8392
Epoch 3/5
1875/1875 [==============================] - 11s 6ms/step - loss: 0.4219 -
accuracy: 0.8498 - val_loss: 0.4496 - val_accuracy: 0.8411
Epoch 4/5
1875/1875 [==============================] - 11s 6ms/step - loss: 0.4134 -
accuracy: 0.8525 - val_loss: 0.4409 - val_accuracy: 0.8431
Epoch 5/5
1875/1875 [==============================] - 11s 6ms/step - loss: 0.4054 -
accuracy: 0.8543 - val_loss: 0.4324 - val_accuracy: 0.8432
Epoch 1/5
1875/1875 [==============================] - 12s 7ms/step - loss: 0.1936 -
accuracy: 0.9266 - val_loss: 0.2835 - val_accuracy: 0.8998
Epoch 2/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.1847 -
accuracy: 0.9300 - val_loss: 0.2866 - val_accuracy: 0.8923
Epoch 3/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.1757 -
accuracy: 0.9338 - val_loss: 0.2758 - val_accuracy: 0.9043
Epoch 4/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.1698 -
accuracy: 0.9348 - val_loss: 0.2675 - val_accuracy: 0.9076
Epoch 5/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.1585 -
accuracy: 0.9395 - val_loss: 0.3083 - val_accuracy: 0.9027
Epoch 1/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.5835 -
accuracy: 0.8406 - val_loss: 0.6044 - val_accuracy: 0.8305
```

```
Epoch 2/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.5761 -
accuracy: 0.8424 - val_loss: 0.6088 - val_accuracy: 0.8311
Epoch 3/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.5711 -
accuracy: 0.8422 - val_loss: 0.5853 - val_accuracy: 0.8348
Epoch 4/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.5672 -
accuracy: 0.8432 - val_loss: 0.5952 - val_accuracy: 0.8289
Epoch 5/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.5617 -
accuracy: 0.8453 - val_loss: 0.5736 - val_accuracy: 0.8421
Epoch 1/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.6943 -
accuracy: 0.8044 - val_loss: 0.6887 - val_accuracy: 0.8052
Epoch 2/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.6880 -
accuracy: 0.8068 - val_loss: 0.7012 - val_accuracy: 0.8042
Epoch 3/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.6825 -
accuracy: 0.8077 - val_loss: 0.6838 - val_accuracy: 0.8117
Epoch 4/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.6745 -
accuracy: 0.8116 - val_loss: 0.6850 - val_accuracy: 0.8096
Epoch 5/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.6732 -
accuracy: 0.8117 - val_loss: 0.6875 - val_accuracy: 0.8041
Epoch 1/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.9532 -
accuracy: 0.7557 - val_loss: 0.9944 - val_accuracy: 0.7363
Epoch 2/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.9509 -
accuracy: 0.7559 - val_loss: 0.9849 - val_accuracy: 0.7388
Epoch 3/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.9472 -
accuracy: 0.7577 - val_loss: 0.9617 - val_accuracy: 0.7558
Epoch 4/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.9447 -
accuracy: 0.7582 - val_loss: 0.9706 - val_accuracy: 0.7420
Epoch 5/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.9395 -
accuracy: 0.7586 - val_loss: 0.9517 - val_accuracy: 0.7542
Epoch 1/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.9666 -
accuracy: 0.7570 - val_loss: 0.9878 - val_accuracy: 0.7445
Epoch 2/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.9620 -
accuracy: 0.7594 - val_loss: 0.9969 - val_accuracy: 0.7372
```

```
Epoch 3/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.9580 -
accuracy: 0.7588 - val_loss: 0.9778 - val_accuracy: 0.7507
Epoch 4/5
1875/1875 [==============================] - 11s 6ms/step - loss: 0.9529 -
accuracy: 0.7599 - val_loss: 0.9652 - val_accuracy: 0.7493
Epoch 5/5
1875/1875 [==============================] - 11s 6ms/step - loss: 0.9487 -
accuracy: 0.7610 - val_loss: 0.9581 - val_accuracy: 0.7573
Epoch 1/5
1875/1875 [==============================] - 10s 5ms/step - loss: 0.3979 -
accuracy: 0.8568 - val_loss: 0.4304 - val_accuracy: 0.8417
Epoch 2/5
1875/1875 [==============================] - 11s 6ms/step - loss: 0.3910 -
accuracy: 0.8590 - val_loss: 0.4152 - val_accuracy: 0.8493
Epoch 3/5
1875/1875 [==============================] - 11s 6ms/step - loss: 0.3872 -
accuracy: 0.8610 - val_loss: 0.4221 - val_accuracy: 0.8470
Epoch 4/5
1875/1875 [==============================] - 10s 6ms/step - loss: 0.3817 -
accuracy: 0.8627 - val_loss: 0.4183 - val_accuracy: 0.8498
Epoch 5/5
1875/1875 [==============================] - 10s 5ms/step - loss: 0.3765 -
accuracy: 0.8643 - val_loss: 0.4078 - val_accuracy: 0.8531
```

Number of parameters for the models

[13]:

```python
# modelLeNet = load_model('modelLeNet.h5')
# modelLeNetReguL2v1 = load_model('modelLeNetReguL2v1.h5')
# modelLeNetReguL2v2 = load_model('modelLeNetReguL2v2.h5')
# modelLeNetReguL1v1 = load_model('modelLeNetReguL1v1.h5')
# modelLeNetReguL1v2 = load_model('modelLeNetReguL1v2.h5')
# modelLeNetGAP = load_model('modelLeNetGAP.h5')


print('Parameters of modelLeNet: {:.4f}'.format(modelLeNet.count_params()))
print('Parameters of modelLeNetReguL2v1: {:.4f}'.format(modelLeNetReguL2v1.
 ↪count_params()))
print('Parameters of modelLeNetReguL2v2: {:.4f}'.format(modelLeNetReguL2v2.
 ↪count_params()))
print('Parameters of modelLeNetReguL1v1: {:.4f}'.format(modelLeNetReguL1v1.
 ↪count_params()))
print('Parameters of modelLeNetReguL1v2: {:.4f}'.format(modelLeNetReguL1v2.
 ↪count_params()))
print('Parameters of modelLeNetGAP: {:.4f}'.format(modelLeNetGAP.
 ↪count_params()))
```

```
Parameters of modelLeNet: 61706.0000
Parameters of modelLeNetReguL2v1: 61706.0000
Parameters of modelLeNetReguL2v2: 61706.0000
Parameters of modelLeNetReguL1v1: 61706.0000
Parameters of modelLeNetReguL1v2: 61706.0000
Parameters of modelLeNetGAP: 2742.0000
```

Summary Table

[14]:

```
modelLeNet.summary()
modelLeNetReguL2v1.summary()
modelLeNetReguL2v2.summary()
modelLeNetReguL1v1.summary()
modelLeNetReguL1v2.summary()
modelLeNetGAP.summary()
```

```
Model: "le_net"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 6)         156
_________________________________________________________________
average_pooling2d (AveragePo (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 10, 10, 16)        2416
_________________________________________________________________
average_pooling2d_1 (Average (None, 5, 5, 16)          0
_________________________________________________________________
flatten (Flatten)            (None, 400)               0
_________________________________________________________________
dense (Dense)                (None, 120)               48120
_________________________________________________________________
dense_1 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_2 (Dense)              (None, 10)                850
=================================================================
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
_________________________________________________________________
Model: "le_net_regu_l2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 28, 28, 6)         156
_________________________________________________________________
average_pooling2d_2 (Average (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 10, 10, 16)        2416
```

```
_________________________________________________________________
average_pooling2d_3 (Average (None, 5, 5, 16)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 400)               0
_________________________________________________________________
dense_3 (Dense)              (None, 120)               48120
_________________________________________________________________
dense_4 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_5 (Dense)              (None, 10)                850
=================================================================
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
_________________________________________________________________
Model: "le_net_regu_l2_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 28, 28, 6)         156
_________________________________________________________________
average_pooling2d_4 (Average (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 10, 10, 16)        2416
_________________________________________________________________
average_pooling2d_5 (Average (None, 5, 5, 16)          0
_________________________________________________________________
flatten_2 (Flatten)          (None, 400)               0
_________________________________________________________________
dense_6 (Dense)              (None, 120)               48120
_________________________________________________________________
dense_7 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_8 (Dense)              (None, 10)                850
=================================================================
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
_________________________________________________________________
Model: "le_net_regu_l1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 28, 28, 6)         156
_________________________________________________________________
average_pooling2d_6 (Average (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 10, 10, 16)        2416
```

```
_________________________________________________________________
average_pooling2d_7 (Average (None, 5, 5, 16)          0
_________________________________________________________________
flatten_3 (Flatten)          (None, 400)               0
_________________________________________________________________
dense_9 (Dense)              (None, 120)               48120
_________________________________________________________________
dense_10 (Dense)             (None, 84)                10164
_________________________________________________________________
dense_11 (Dense)             (None, 10)                850
=================================================================
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
_________________________________________________________________
Model: "le_net_regu_l1_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_8 (Conv2D)            (None, 28, 28, 6)         156
_________________________________________________________________
average_pooling2d_8 (Average (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 10, 10, 16)        2416
_________________________________________________________________
average_pooling2d_9 (Average (None, 5, 5, 16)          0
_________________________________________________________________
flatten_4 (Flatten)          (None, 400)               0
_________________________________________________________________
dense_12 (Dense)             (None, 120)               48120
_________________________________________________________________
dense_13 (Dense)             (None, 84)                10164
_________________________________________________________________
dense_14 (Dense)             (None, 10)                850
=================================================================
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
_________________________________________________________________
Model: "le_net_gap"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_10 (Conv2D)           (None, 28, 28, 6)         156
_________________________________________________________________
average_pooling2d_10 (Averag (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 10, 10, 16)        2416
```

```
_________________________________________________________________
average_pooling2d_11 (Averag (None, 5, 5, 16)          0
_________________________________________________________________
global_max_pooling2d (Global (None, 16)                0
_________________________________________________________________
dense_15 (Dense)             (None, 10)                170
=================================================================
Total params: 2,742
Trainable params: 2,742
Non-trainable params: 0
_________________________________________________________________
```

[15]:

```
# Table
model_info =␣
 ↪['LeNet','LeNetReguL2v1','LeNetReguL2v2','LeNetReguL1v1','LeNetReguL1v2',␣
 ↪'LeNetGAP']
print('Model\t\t\tTraining Error\t\t\tTest Error\t\t\tSD Of Test␣
 ↪Error\t\t\tInference Time\t\t\tNo. of Parameters')
print('-----------------\t-----------------\t\t-----------------\t\t-----------------\t\t\t---
print(str(model_info[0]) + '\t\t\t' + str(model_info_LeNet['TrainingError']) +␣
 ↪'\t\t' + str(model_info_LeNet['TestError'])+ '\t\t' + str(np.
 ↪std(model_info_LeNet['SDTestError']))+ '\t\t\t' +␣
 ↪str(model_info_LeNet['InferenceTime'])+ '\t\t' +␣
 ↪str(model_info_LeNet['NoofParameters']))
print(str(model_info[1]) + '\t\t' +␣
 ↪str(model_info_LeNetReguL2v1['TrainingError']) + '\t\t' +␣
 ↪str(model_info_LeNetReguL2v1['TestError'])+ '\t\t' + str(np.
 ↪std(model_info_LeNetReguL2v1['SDTestError']))+ '\t\t\t' +␣
 ↪str(model_info_LeNetReguL2v1['InferenceTime'])+ '\t\t' +␣
 ↪str(model_info_LeNetReguL2v1['NoofParameters']))
print(str(model_info[2]) + '\t\t' +␣
 ↪str(model_info_LeNetReguL2v2['TrainingError']) + '\t\t' +␣
 ↪str(model_info_LeNetReguL2v2['TestError'])+ '\t\t' + str(np.
 ↪std(model_info_LeNetReguL2v2['SDTestError']))+ '\t\t\t' +␣
 ↪str(model_info_LeNetReguL2v2['InferenceTime'])+ '\t\t' +␣
 ↪str(model_info_LeNetReguL2v2['NoofParameters']))
print(str(model_info[3]) + '\t\t' +␣
 ↪str(model_info_LeNetReguL1v1['TrainingError']) + '\t\t' +␣
 ↪str(model_info_LeNetReguL1v1['TestError'])+ '\t\t' + str(np.
 ↪std(model_info_LeNetReguL1v1['SDTestError']))+ '\t\t\t' +␣
 ↪str(model_info_LeNetReguL1v1['InferenceTime'])+ '\t\t' +␣
 ↪str(model_info_LeNetReguL1v1['NoofParameters']))
```

```
print(str(model_info[4]) + '\t\t' +␣
 ↪str(model_info_LeNetReguL1v2['TrainingError']) + '\t\t' +␣
 ↪str(model_info_LeNetReguL1v2['TestError'])+ '\t\t' + str(np.
 ↪std(model_info_LeNetReguL1v2['SDTestError']))+ '\t\t\t' +␣
 ↪str(model_info_LeNetReguL1v2['InferenceTime'])+ '\t\t' +␣
 ↪str(model_info_LeNetReguL1v2['NoofParameters']))
print(str(model_info[5]) + '\t\t' + str(model_info_LeNetGAP['TrainingError']) +␣
 ↪'\t\t' + str(model_info_LeNetGAP['TestError'])+ '\t\t' + str(np.
 ↪std(model_info_LeNetGAP['SDTestError']))+ '\t\t\t' +␣
 ↪str(model_info_LeNetGAP['InferenceTime'])+ '\t\t' +␣
 ↪str(model_info_LeNetGAP['NoofParameters']))
```

```
Model                   Training Error                  Test Error
SD Of Test Error                        Inference Time                  No. of
Parameters
-----------------       -----------------               -----------------
-----------------                       -----------------
-----------------
LeNet                   0.15851141512393951             0.3083064556121826
0.007505538223938842                    0.8528482913970947              61706
LeNetReguL2v1           0.5616891980171204              0.5736199021339417
0.03166930121361471                     0.878307580947876               61706
LeNetReguL2v2           0.6731531620025635              0.6875057816505432
0.040464484442225626                    0.8369247913360596              61706
LeNetReguL1v1           0.9395244717597961              0.9516729712486267
0.02601441622465937                     0.8399224281311035              61706
LeNetReguL1v2           0.9486860632896423              0.9581499099731445
0.025203893592643113                    0.7941558361053467              61706
LeNetGAP                0.37653425335884094             0.4078139066696167
0.02924511829714039                     0.7626855373382568              2742
```

Sparsity of layers

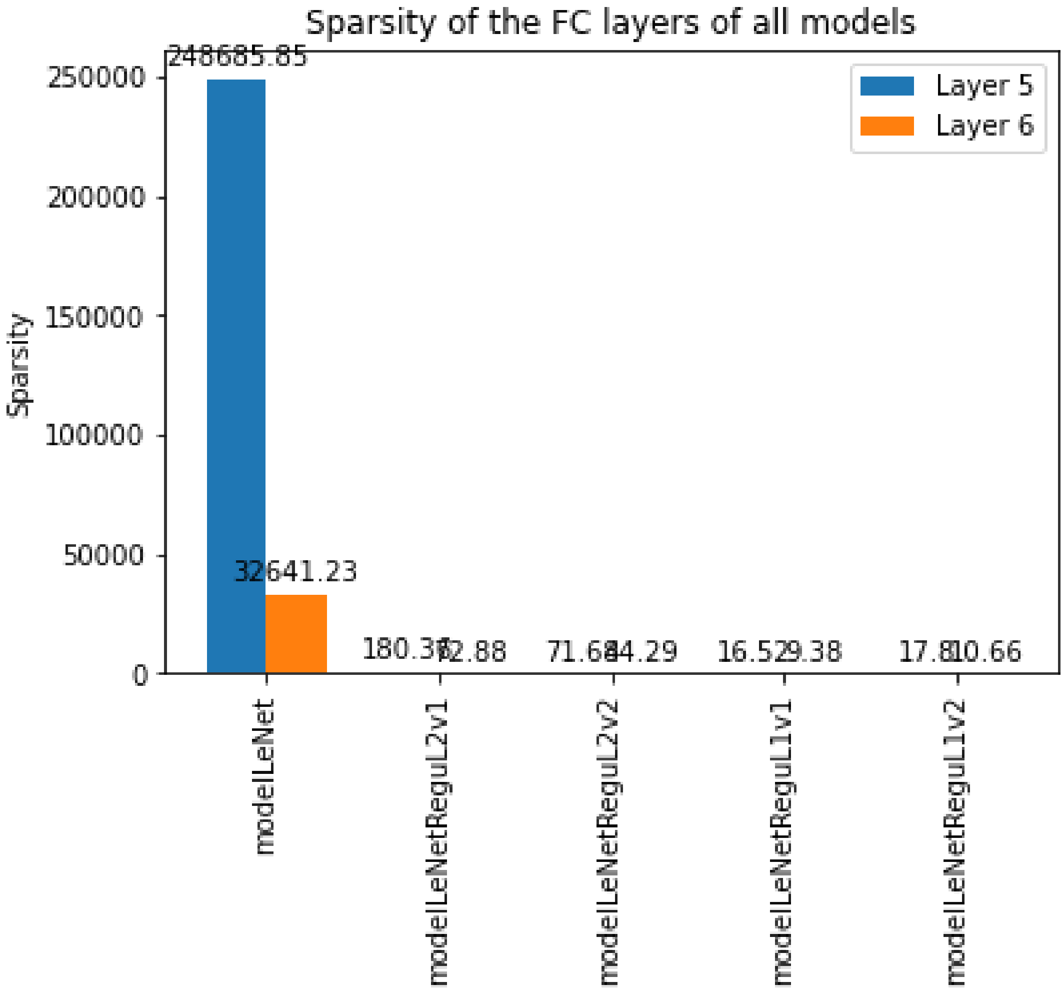
[16]:

```
l5modelLeNet = round(hoyer_index(modelLeNet.layers[5].get_weights()[0]),2)
l6modelLeNet = round(hoyer_index(modelLeNet.layers[6].get_weights()[0]),2)

l5modelLeNetReguL2v1 = round(hoyer_index(modelLeNetReguL2v1.layers[5].
 ↪get_weights()[0]),2)
l6modelLeNetReguL2v1 = round(hoyer_index(modelLeNetReguL2v1.layers[6].
 ↪get_weights()[0]),2)

l5modelLeNetReguL2v2 = round(hoyer_index(modelLeNetReguL2v2.layers[5].
 ↪get_weights()[0]),2)
l6modelLeNetReguL2v2 = round(hoyer_index(modelLeNetReguL2v2.layers[6].
 ↪get_weights()[0]),2)
```

```
l5modelLeNetReguL1v1 = round(hoyer_index(modelLeNetReguL1v1.layers[5].
 ↪get_weights()[0]),2)
l6modelLeNetReguL1v1 = round(hoyer_index(modelLeNetReguL1v1.layers[6].
 ↪get_weights()[0]),2)

l5modelLeNetReguL1v2 = round(hoyer_index(modelLeNetReguL1v2.layers[5].
 ↪get_weights()[0]),2)
l6modelLeNetReguL1v2 = round(hoyer_index(modelLeNetReguL1v2.layers[6].
 ↪get_weights()[0]),2)

print('Sparsity of layer 5 - modelLeNet: {:.2f}'.format(l5modelLeNet))
print('Sparsity of layer 6 - modelLeNet: {:.2f}'.format(l6modelLeNet))

print('Sparsity of layer 5 - modelLeNetReguL2v1: {:.2f}'.
 ↪format(l5modelLeNetReguL2v1))
print('Sparsity of layer 6 - modelLeNetReguL2v1: {:.2f}'.
 ↪format(l6modelLeNetReguL2v1))

print('Sparsity of layer 5 - modelLeNetReguL2v2: {:.2f}'.
 ↪format(l5modelLeNetReguL2v2))
print('Sparsity of layer 6 - modelLeNetReguL2v2: {:.2f}'.
 ↪format(l6modelLeNetReguL2v2))

print('Sparsity of layer 5 - modelLeNetReguL1v1: {:.2f}'.
 ↪format(l5modelLeNetReguL1v1))
print('Sparsity of layer 6 - modelLeNetReguL1v1: {:.2f}'.
 ↪format(l6modelLeNetReguL1v1))

print('Sparsity of layer 5 - modelLeNetReguL1v2: {:.2f}'.
 ↪format(l5modelLeNetReguL1v2))
print('Sparsity of layer 6 - modelLeNetReguL1v2: {:.2f}'.
 ↪format(l6modelLeNetReguL1v2))


import matplotlib
import matplotlib.pyplot as plt
import numpy as np

labels = ['modelLeNet', 'modelLeNetReguL2v1', 'modelLeNetReguL2v2',␣
 ↪'modelLeNetReguL1v1', 'modelLeNetReguL1v2']
layer_5 = [l5modelLeNet, l5modelLeNetReguL2v1, l5modelLeNetReguL2v2,␣
 ↪l5modelLeNetReguL1v1, l5modelLeNetReguL1v2]
layer_6 = [l6modelLeNet, l6modelLeNetReguL2v1, l6modelLeNetReguL2v2,␣
 ↪l6modelLeNetReguL1v1, l6modelLeNetReguL1v2]

# locations of the labels
```

```python
x = np.arange(len(labels))
# set the width of each element of the group
width = 0.35

fig, ax = plt.subplots()
rects1 = ax.bar(x - width/2, layer_5, width, label='Layer 5')
rects2 = ax.bar(x + width/2, layer_6, width, label='Layer 6')

# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_ylabel('Sparsity')
ax.set_title('Sparsity of the FC layers of all models')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend()

def autolabel(rects):
    """Sparcity above each bar."""
    for rect in rects:
        height = rect.get_height()
        ax.annotate('{}'.format(height),
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 3),
                    textcoords="offset points",
                    ha='center', va='bottom')


autolabel(rects1)
autolabel(rects2)
fig.tight_layout()
plt.xticks(rotation=90)
plt.show()

# Reference: https://matplotlib.org/3.1.1/gallery/lines_bars_and_markers/
 ↪barchart.html
```

```
Sparsity of layer 5 - modelLeNet: 248685.85
Sparsity of layer 6 - modelLeNet: 32641.23
Sparsity of layer 5 - modelLeNetReguL2v1: 180.36
Sparsity of layer 6 - modelLeNetReguL2v1: 72.88
Sparsity of layer 5 - modelLeNetReguL2v2: 71.68
Sparsity of layer 6 - modelLeNetReguL2v2: 44.29
Sparsity of layer 5 - modelLeNetReguL1v1: 16.52
Sparsity of layer 6 - modelLeNetReguL1v1: 9.38
Sparsity of layer 5 - modelLeNetReguL1v2: 17.80
Sparsity of layer 6 - modelLeNetReguL1v2: 10.66
```



```
[ ]:
```

Board

```
[17]: %load_ext tensorboard
      %tensorboard --logdir logs/fit
```

```
Reusing TensorBoard on port 6006 (pid 21495), started 7:43:59 ago. (Use '!kill 21495' to kill
```