Practice-CNN

April 26, 2023

[1]:
```
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784', version=1)
mnist.keys()
```

```
/Users/istiqomah/miniforge3/envs/tensorEnv/lib/python3.8/site-
packages/sklearn/datasets/_openml.py:968: FutureWarning: The default value of
`parser` will change from `'liac-arff'` to `'auto'` in 1.4. You can set
`parser='auto'` to silence this warning. Therefore, an `ImportError` will be
raised from 1.4 if the dataset is dense and pandas is not installed. Note that
the pandas parser may return different data types. See the Notes Section in
fetch_openml's API doc for details.
  warn(
```

[1]:
```
dict_keys(['data', 'target', 'frame', 'categories', 'feature_names',
'target_names', 'DESCR', 'details', 'url'])
```

[19]:
```
X, y = mnist["data"], mnist["target"]
X.shape
```

[19]:
```
(70000, 784)
```

[20]:
```
import numpy as np
import tensorflow as tf
from keras.models import Model
from keras.layers import Input, Activation, Dense, Conv2D, MaxPooling2D,␣
 ↪ZeroPadding2D, Flatten, Dropout
from keras.optimizers import Adam
from keras.utils.np_utils import to_categorical
from keras.callbacks import TensorBoard
import numpy as np
from sklearn.model_selection import train_test_split
X= X.to_numpy()
y = to_categorical( y )
X_data, y_data = X[:60000,:], y[:60000]
X_test, y_test = X[60000:,:], y[60000:]
X_train, y_train = X_data[:50000,:], y_data[:50000]
X_valid, y_valid = X_data[50000:,:], y_data[50000:]
```

```
print('Training:   ', X_train.shape, y_train.shape)
print('Validation: ', X_valid.shape, y_valid.shape)
print('Test Set:   ', X_test.shape, y_test.shape)

X_train = np.reshape(X_train, (len(X_train), 28, 28, 1))
X_valid = np.reshape(X_valid, (len(X_valid), 28, 28, 1))
X_test = np.reshape(X_test, (len(X_test), 28, 28, 1))
```

```
Training:    (50000, 784) (50000, 10)
Validation:  (10000, 784) (10000, 10)
Test Set:    (10000, 784) (10000, 10)
```

[4]:
```
# Feature Extraction Layer
inputs = Input(shape=(28, 28, 1))
conv_layer = ZeroPadding2D(padding=(2,2))(inputs)
conv_layer = Conv2D(32, (5, 5), strides=(1,1), activation='relu')(conv_layer)
conv_layer = MaxPooling2D((2, 2))(conv_layer)
conv_layer = Conv2D(64, (5, 5), strides=(1,1), activation='relu')(conv_layer)
conv_layer = MaxPooling2D((2, 2))(conv_layer)

# Flatten feature map to Vector with 576 element.
flatten = Flatten()(conv_layer)

# Fully Connected Layer
fc_layer = Dense(1024, activation='relu')(flatten)
fc_layer = Dropout(0.5)(fc_layer)
outputs = Dense(10, activation='softmax')(fc_layer)

model = Model(inputs=inputs, outputs=outputs)

# Adam Optimizer and Cross Entropy Loss
adam = Adam(lr=0.0001)
model.compile(optimizer=adam, loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])

# Print Model Summary
print(model.summary())
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 28, 28, 1)]       0

 zero_padding2d (ZeroPadding  (None, 32, 32, 1)        0
 2D)

 conv2d (Conv2D)             (None, 28, 28, 32)        832
```

```
max_pooling2d (MaxPooling2D  (None, 14, 14, 32)       0
)

conv2d_1 (Conv2D)            (None, 10, 10, 64)       51264

max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
2D)

flatten (Flatten)            (None, 1600)             0

dense (Dense)                (None, 1024)             1639424

dropout (Dropout)            (None, 1024)             0

dense_1 (Dense)              (None, 10)               10250

=================================================================
Total params: 1,701,770
Trainable params: 1,701,770
Non-trainable params: 0
_________________________________________________________________
None
```

```
/Users/istiqomah/miniforge3/envs/tensorEnv/lib/python3.8/site-
packages/keras/optimizers/legacy/adam.py:117: UserWarning: The `lr` argument is
deprecated, use `learning_rate` instead.
  super().__init__(name, **kwargs)
```

[5]:
```
# Use TensorBoard
callbacks = TensorBoard(log_dir='./Graph')

# Train for 100 Epochs and use TensorBoard Callback
model.fit(X_train, y_train, batch_size=64, epochs=20, verbose=1,␣
 ↪validation_data=(X_valid, y_valid), callbacks=[callbacks])

# Save Weights
model.save_weights('weights.h5')
```

```
Epoch 1/20
782/782 [==============================] - 14s 17ms/step - loss: 1.2008 -
accuracy: 0.8796 - val_loss: 0.1147 - val_accuracy: 0.9738
Epoch 2/20
782/782 [==============================] - 13s 17ms/step - loss: 0.1440 -
accuracy: 0.9620 - val_loss: 0.0837 - val_accuracy: 0.9774
Epoch 3/20
782/782 [==============================] - 13s 17ms/step - loss: 0.0823 -
accuracy: 0.9762 - val_loss: 0.0603 - val_accuracy: 0.9844
```

```
Epoch 4/20
782/782 [==============================] - 13s 17ms/step - loss: 0.0564 -
accuracy: 0.9825 - val_loss: 0.0513 - val_accuracy: 0.9868
Epoch 5/20
782/782 [==============================] - 13s 17ms/step - loss: 0.0415 -
accuracy: 0.9872 - val_loss: 0.0493 - val_accuracy: 0.9855
Epoch 6/20
782/782 [==============================] - 13s 16ms/step - loss: 0.0333 -
accuracy: 0.9892 - val_loss: 0.0524 - val_accuracy: 0.9876
Epoch 7/20
782/782 [==============================] - 13s 17ms/step - loss: 0.0280 -
accuracy: 0.9907 - val_loss: 0.0472 - val_accuracy: 0.9883
Epoch 8/20
782/782 [==============================] - 13s 16ms/step - loss: 0.0211 -
accuracy: 0.9927 - val_loss: 0.0565 - val_accuracy: 0.9870
Epoch 9/20
782/782 [==============================] - 13s 16ms/step - loss: 0.0192 -
accuracy: 0.9936 - val_loss: 0.0525 - val_accuracy: 0.9882
Epoch 10/20
782/782 [==============================] - 13s 17ms/step - loss: 0.0163 -
accuracy: 0.9944 - val_loss: 0.0508 - val_accuracy: 0.9894
Epoch 11/20
782/782 [==============================] - 13s 16ms/step - loss: 0.0149 -
accuracy: 0.9950 - val_loss: 0.0479 - val_accuracy: 0.9906
Epoch 12/20
782/782 [==============================] - 12s 16ms/step - loss: 0.0124 -
accuracy: 0.9960 - val_loss: 0.0567 - val_accuracy: 0.9875
Epoch 13/20
782/782 [==============================] - 13s 17ms/step - loss: 0.0100 -
accuracy: 0.9963 - val_loss: 0.0476 - val_accuracy: 0.9897
Epoch 14/20
782/782 [==============================] - 13s 17ms/step - loss: 0.0100 -
accuracy: 0.9965 - val_loss: 0.0564 - val_accuracy: 0.9885
Epoch 15/20
782/782 [==============================] - 14s 18ms/step - loss: 0.0122 -
accuracy: 0.9962 - val_loss: 0.0564 - val_accuracy: 0.9891
Epoch 16/20
782/782 [==============================] - 14s 17ms/step - loss: 0.0092 -
accuracy: 0.9969 - val_loss: 0.0562 - val_accuracy: 0.9889
Epoch 17/20
782/782 [==============================] - 13s 17ms/step - loss: 0.0051 -
accuracy: 0.9982 - val_loss: 0.0564 - val_accuracy: 0.9893
Epoch 18/20
782/782 [==============================] - 13s 17ms/step - loss: 0.0064 -
accuracy: 0.9977 - val_loss: 0.0495 - val_accuracy: 0.9905
Epoch 19/20
782/782 [==============================] - 13s 17ms/step - loss: 0.0078 -
accuracy: 0.9974 - val_loss: 0.0581 - val_accuracy: 0.9904
```

```
Epoch 20/20
782/782 [==============================] - 13s 17ms/step - loss: 0.0066 -
accuracy: 0.9979 - val_loss: 0.0510 - val_accuracy: 0.9909
```

[22]:
```
model.evaluate(X_test, y_test, verbose=0)
```

[22]:
```
[0.04141769930720329, 0.9904000759124756]
```

[24]:
```
pred = model.predict(X_test[0].reshape(1, 28, 28, 1))
print(pred.argmax())
print(y_test[0].argmax())
```

```
1/1 [==============================] - 0s 22ms/step
7
7
```