mnist_optuna(1)

March 12, 2021

```
[ ]: !pip install optuna
```

```
[ ]: import tensorflow as tf
     import tensorflow.keras as keras
     from keras.datasets import mnist
     from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D,␣
      ↪BatchNormalization, Activation, AveragePooling2D
     import tensorflow.keras.backend as K
     from tensorflow.keras import Sequential

     import optuna
```

```
[ ]: batch_size = 128
     num_classes = 10
     epochs = 12
```

```
[ ]: img_rows, img_cols = 28, 28
     (x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
[ ]: x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
     x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
```

```
[ ]: x_train = x_train.astype('float32')
     x_test = x_test.astype('float32')
     x_train = x_train / 255.0
     x_test = x_test / 255.0
```

```
[ ]: y_train = keras.utils.to_categorical(y_train, num_classes)
     y_test = keras.utils.to_categorical(y_test, num_classes)
```

```
[ ]: params = {
         'conv_filters': [[16, 64], [16, 128]],
         'lconv_filters': [32, 128]
     }
```

```
[ ]: def define_model(trial):
    model_full_cnn = Sequential()

    filter_size = trial.suggest_int('conv'+str(1),␣
↪params['conv_filters'][0][0], params['conv_filters'][0][1])
    model_full_cnn.add(Conv2D(filters=filter_size, kernel_size=(3, 3),␣
↪padding='same', input_shape=(28, 28, 1),
                              use_bias=False))
    model_full_cnn.add(BatchNormalization())
    model_full_cnn.add(Activation('relu'))
    model_full_cnn.add(MaxPooling2D(pool_size=(2, 2)))
    p = trial.suggest_float('conv_drop'+str(1), 0.0, 0.5)
    model_full_cnn.add(Dropout(rate=p))

    filter_size = trial.suggest_int('conv'+str(2),␣
↪params['conv_filters'][1][0], params['conv_filters'][1][1])
    model_full_cnn.add(Conv2D(filters=filter_size, kernel_size=(3, 3),␣
↪padding='same', input_shape=(28, 28, 1),
                              use_bias=False))
    model_full_cnn.add(BatchNormalization())
    model_full_cnn.add(Activation('relu'))
    model_full_cnn.add(MaxPooling2D(pool_size=(2, 2)))
    p = trial.suggest_float('conv_drop'+str(2), 0.0, 0.5)
    model_full_cnn.add(Dropout(rate=p))

    filter_size = trial.suggest_int('lconv'+str(1), params['lconv_filters'][0],␣
↪params['lconv_filters'][1])
    model_full_cnn.add(Conv2D(filters=filter_size, kernel_size=(3, 3),␣
↪padding='same'))
    filter_size = trial.suggest_int('lconv'+str(2), params['lconv_filters'][0],␣
↪params['lconv_filters'][1])
    model_full_cnn.add(Conv2D(filters=filter_size, kernel_size=(3, 3),␣
↪padding='same', use_bias=False))

    model_full_cnn.add(BatchNormalization())
    model_full_cnn.add(Activation('relu'))
    model_full_cnn.add(MaxPooling2D(pool_size=(2, 2)))
    p = trial.suggest_float('lconv_drop'+str(1), 0.0, 0.5)
    model_full_cnn.add(Dropout(rate=p))

    model_full_cnn.add(Conv2D(filters=10, kernel_size=(1, 1), padding='same'))
    model_full_cnn.add(AveragePooling2D(pool_size=(3, 3)))
    model_full_cnn.add(Flatten())
    model_full_cnn.add(Activation('softmax'))


    return model_full_cnn
```

```
[ ]: class OptunaReporter(keras.callbacks.Callback):

         def __init__(self, trial):
             self.trial = trial

         def on_epoch_end(self, epoch, logs=None):
             self.trial.report(logs['accuracy'], epoch)
```

```
[ ]: def objective(trial):
         lr = trial.suggest_float('lr', 1e-5, 1e-1, log=True)
         optimizer = keras.optimizers.Adam(learning_rate=lr)
         model = define_model(trial)
         model.compile(loss=keras.losses.categorical_crossentropy,␣
      ↪optimizer=optimizer, metrics=['accuracy'])
         model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs,␣
      ↪verbose=0, validation_data=(x_test, y_test),␣
      ↪callbacks=[OptunaReporter(trial)])
         score = model.evaluate(x_test, y_test, verbose=0)

         if trial.should_prune():
             raise optuna.exceptions.TrialPruned()

         return score[1]
```

```
[ ]: study = optuna.create_study(direction='maximize')
     study.optimize(objective, n_trials=100)
```

```
[ ]: study.best_params, study.best_value
```

```
[ ]: ({'conv1': 53,
       'conv2': 50,
       'conv_drop1': 0.24675468866323005,
       'conv_drop2': 0.24837435048210701,
       'lconv1': 108,
       'lconv2': 47,
       'lconv_drop1': 0.0257626138215777,
       'lr': 0.0011600392998574705},
      0.9934999942779541)
```

```
[ ]: fig = optuna.visualization.plot_param_importances(study)
     fig.show()
```

```
[ ]:
```