02_digit_classification_with_lenet5

September 29, 2021

1 Basic Image Classification with Feedforward NN and LetNet5

All libraries we introduced in the last chapter provide support for convolutional layers. We are going to illustrate the LeNet5 architecture using the most basic MNIST handwritten digit dataset, and then use AlexNet on CIFAR10, a simplified version of the original ImageNet to demonstrate the use of data augmentation. LeNet5 and MNIST using Keras.

1.1 Imports

[1]:
```
%matplotlib inline

from pathlib import Path
from random import randint
import numpy as np
import pandas as pd

import tensorflow as tf
from tensorflow.keras.datasets import mnist
import tensorflow.keras.backend as K
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import (Conv2D,
                                     AveragePooling2D,
                                     Dense,
                                     Dropout,
                                     Flatten)
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
gpu_devices = tf.config.experimental.list_physical_devices('GPU')
if gpu_devices:
    print('Using GPU')
    tf.config.experimental.set_memory_growth(gpu_devices[0], True)
else:
    print('Using CPU')
```

```
Using CPU
```

[3]:
```
sns.set_style('whitegrid')
```

[4]:
```
results_path = Path('results')
mnist_path = results_path / 'mnist'
if not mnist_path.exists():
    mnist_path.mkdir(parents=True)
```

1.2 Load MNIST Database

The original MNIST dataset contains 60,000 images in 28x28 pixel resolution with a single grayscale containing handwritten digits from 0 to 9. A good alternative is the more challenging but structurally similar Fashion MNIST dataset that we encountered in Chapter 12 on Unsupervised Learning.

We can load it in keras out of the box:

[5]:
```
# use Keras to import pre-shuffled MNIST database
(X_train, y_train), (X_test, y_test) = mnist.load_data()

print("The MNIST database has a training set of %d examples." % len(X_train))
print("The MNIST database has a test set of %d examples." % len(X_test))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
The MNIST database has a training set of 60000 examples.
The MNIST database has a test set of 10000 examples.
```

[6]:
```
X_train.shape, X_test.shape
```

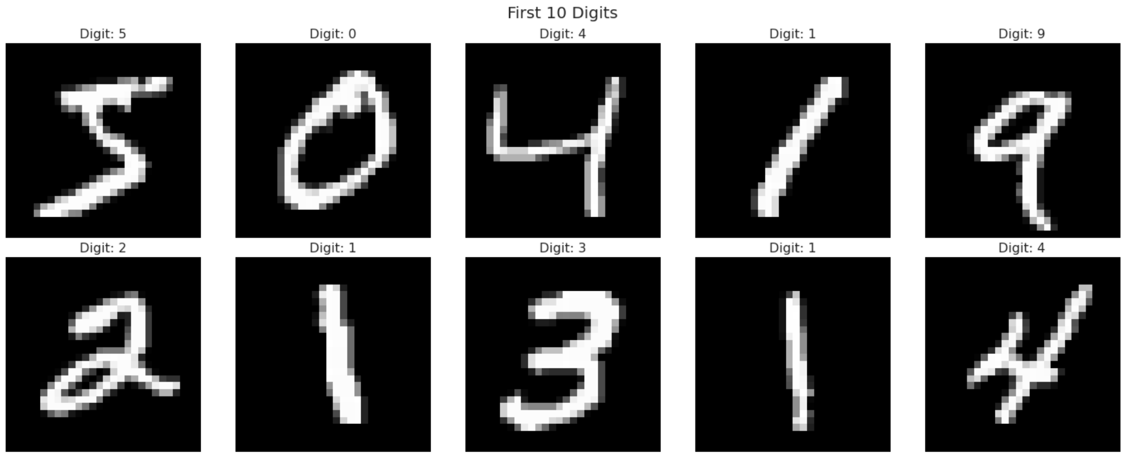
[6]:
```
((60000, 28, 28), (10000, 28, 28))
```

1.3 Visualize Data

1.3.1 Visualize First 10 Training Images

The below figure shows the first ten images in the dataset and highlights significant variation among instances of the same digit. On the right, it shows how the pixel values for an indivual image range from 0 to 255.

[7]:
```
fig, axes = plt.subplots(ncols=5, nrows=2, figsize=(20, 8))
axes = axes.flatten()
for i, ax in enumerate(axes):
    ax.imshow(X_train[i], cmap='gray')
    ax.axis('off')
    ax.set_title('Digit: {}'.format(y_train[i]), fontsize=16)
fig.suptitle('First 10 Digits', fontsize=20)
fig.tight_layout()
fig.subplots_adjust(top=.9)
```



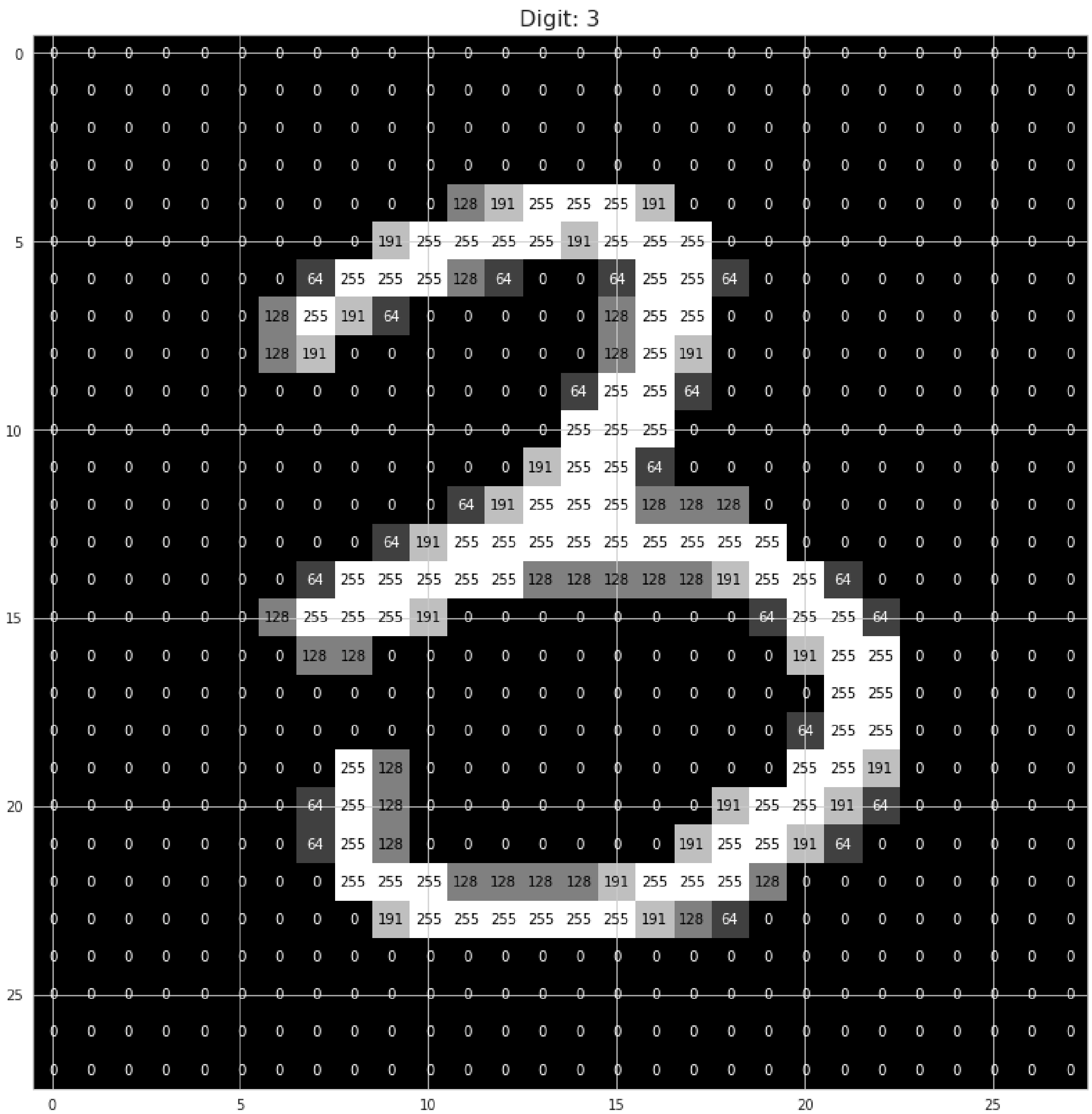
1.3.2 Show random image in detail

[8]:
```
fig, ax = plt.subplots(figsize = (14, 14))

i = randint(0, len(X_train))
img = X_train[i]

ax.imshow(img, cmap='gray')
ax.set_title('Digit: {}'.format(y_train[i]), fontsize=16)

width, height = img.shape
thresh = img.max()/2.5
for x in range(width):
    for y in range(height):
        ax.annotate('{:2}'.format(img[x][y]),
                    xy=(y,x),
                    horizontalalignment='center',
                    verticalalignment='center',
                    color='white' if img[x][y]<thresh else 'black')
```



1.4 Prepare Data

1.4.1 Rescale pixel values

We rescale the pixel values to the range [0, 1] to normalize the training data and faciliate the backpropagation process and convert the data to 32 bit floats that reduce memory requirements and computational cost while providing sufficient precision for our use case:

[9]:
```
# rescale [0,255] --> [0,1]
X_train = X_train.astype('float32')/255
X_test = X_test.astype('float32')/255
```

1.4.2 One-Hot Label Encoding using Keras

Print first ten labels

[10]:
```python
print('Integer-valued labels:')
print(y_train[:10])
```

```
Integer-valued labels:
[5 0 4 1 9 2 1 3 1 4]
```

1.5 Feed-Forward NN

1.5.1 Model Architecture

[11]:
```python
ffnn = Sequential([
    Flatten(input_shape=X_train.shape[1:]),
    Dense(512, activation='relu'),
    Dropout(0.2),
    Dense(512, activation='relu'),
    Dropout(0.2),
    Dense(10, activation='softmax')
])
```

[12]:
```python
ffnn.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense (Dense)                (None, 512)               401920
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                5130
=================================================================
Total params: 669,706
Trainable params: 669,706
Non-trainable params: 0
_________________________________________________________________
```

1.5.2 Compile the Model

[13]:
```
ffnn.compile(loss='sparse_categorical_crossentropy',
             optimizer='rmsprop',
             metrics=['accuracy'])
```

1.5.3 Calculate Baseline Classification Accuracy

[14]:
```
# evaluate test accuracy
baseline_accuracy = ffnn.evaluate(X_test, y_test, verbose=0)[1]

# print test accuracy
print(f'Test accuracy: {baseline_accuracy:.2%}')
```

```
Test accuracy: 13.07%
```

1.5.4 Callback for model persistence

[15]:
```
ffn_path = mnist_path / 'ffn.best.hdf5'
```

[16]:
```
checkpointer = ModelCheckpoint(filepath=ffn_path.as_posix(),
                               verbose=1,
                               save_best_only=True)
```

1.5.5 Early Stopping Callback

[17]:
```
early_stopping = EarlyStopping(monitor='val_loss', patience=20)
```

1.5.6 Train the Model

[18]:
```
epochs = 100
batch_size = 32
```

[19]:
```
ffnn_history = ffnn.fit(X_train,
                        y_train,
                        batch_size=batch_size,
                        epochs=epochs,
                        validation_split=0.2,
                        callbacks=[checkpointer, early_stopping],
                        verbose=1,
                        shuffle=True)
```

```
Epoch 1/100
1499/1500 [============================>.] - ETA: 0s - loss: 0.2358 - accuracy:
0.9301
Epoch 00001: val_loss improved from inf to 0.13914, saving model to
results/mnist/ffn.best.hdf5
```

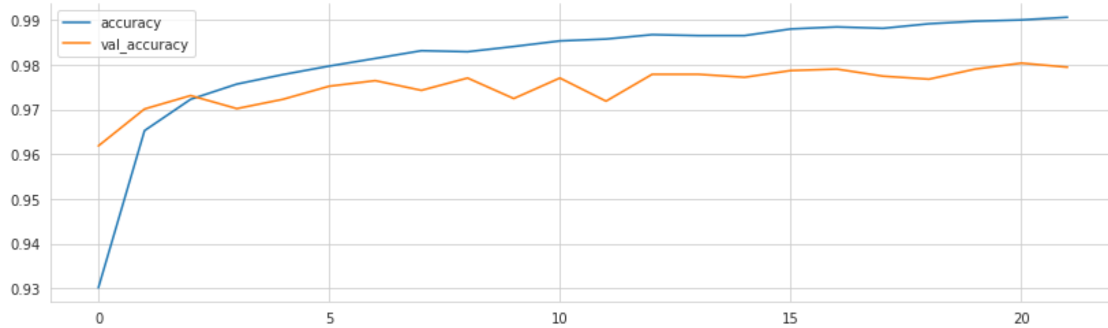
```
1500/1500 [==============================] - 6s 4ms/step - loss: 0.2357 -
accuracy: 0.9301 - val_loss: 0.1391 - val_accuracy: 0.9618
Epoch 2/100
1494/1500 [============================>.] - ETA: 0s - loss: 0.1322 - accuracy:
0.9652
Epoch 00002: val_loss improved from 0.13914 to 0.12946, saving model to
results/mnist/ffn.best.hdf5
1500/1500 [==============================] - 6s 4ms/step - loss: 0.1320 -
accuracy: 0.9653 - val_loss: 0.1295 - val_accuracy: 0.9701
Epoch 3/100
1494/1500 [============================>.] - ETA: 0s - loss: 0.1153 - accuracy:
0.9724
Epoch 00003: val_loss did not improve from 0.12946
1500/1500 [==============================] - 6s 4ms/step - loss: 0.1162 -
accuracy: 0.9722 - val_loss: 0.1344 - val_accuracy: 0.9731
Epoch 4/100
1493/1500 [============================>.] - ETA: 0s - loss: 0.1088 - accuracy:
0.9757
Epoch 00004: val_loss did not improve from 0.12946
1500/1500 [==============================] - 6s 4ms/step - loss: 0.1088 -
accuracy: 0.9756 - val_loss: 0.1794 - val_accuracy: 0.9702
Epoch 5/100
1489/1500 [============================>.] - ETA: 0s - loss: 0.1047 - accuracy:
0.9778
Epoch 00005: val_loss did not improve from 0.12946
1500/1500 [==============================] - 6s 4ms/step - loss: 0.1047 -
accuracy: 0.9778 - val_loss: 0.1506 - val_accuracy: 0.9722
Epoch 6/100
1492/1500 [============================>.] - ETA: 0s - loss: 0.0933 - accuracy:
0.9796
Epoch 00006: val_loss did not improve from 0.12946
1500/1500 [==============================] - 6s 4ms/step - loss: 0.0930 -
accuracy: 0.9797 - val_loss: 0.1600 - val_accuracy: 0.9752
Epoch 7/100
1494/1500 [============================>.] - ETA: 0s - loss: 0.0970 - accuracy:
0.9814
Epoch 00007: val_loss did not improve from 0.12946
1500/1500 [==============================] - 6s 4ms/step - loss: 0.0969 -
accuracy: 0.9814 - val_loss: 0.1674 - val_accuracy: 0.9764
Epoch 8/100
1489/1500 [============================>.] - ETA: 0s - loss: 0.0898 - accuracy:
0.9831
Epoch 00008: val_loss did not improve from 0.12946
1500/1500 [==============================] - 6s 4ms/step - loss: 0.0901 -
accuracy: 0.9831 - val_loss: 0.2152 - val_accuracy: 0.9743
Epoch 9/100
1491/1500 [============================>.] - ETA: 0s - loss: 0.0891 - accuracy:
0.9828
```

```
Epoch 00009: val_loss did not improve from 0.12946
1500/1500 [==============================] - 6s 4ms/step - loss: 0.0887 -
accuracy: 0.9829 - val_loss: 0.2123 - val_accuracy: 0.9770
Epoch 10/100
1488/1500 [============================>.] - ETA: 0s - loss: 0.0899 - accuracy:
0.9841
Epoch 00010: val_loss did not improve from 0.12946
1500/1500 [==============================] - 6s 4ms/step - loss: 0.0903 -
accuracy: 0.9840 - val_loss: 0.2225 - val_accuracy: 0.9724
Epoch 11/100
1498/1500 [============================>.] - ETA: 0s - loss: 0.0800 - accuracy:
0.9853
Epoch 00011: val_loss did not improve from 0.12946
1500/1500 [==============================] - 6s 4ms/step - loss: 0.0799 -
accuracy: 0.9853 - val_loss: 0.2246 - val_accuracy: 0.9770
Epoch 12/100
1500/1500 [==============================] - ETA: 0s - loss: 0.0760 - accuracy:
0.9857
Epoch 00012: val_loss did not improve from 0.12946
1500/1500 [==============================] - 6s 4ms/step - loss: 0.0760 -
accuracy: 0.9857 - val_loss: 0.2523 - val_accuracy: 0.9718
Epoch 13/100
1500/1500 [==============================] - ETA: 0s - loss: 0.0756 - accuracy:
0.9867
Epoch 00013: val_loss did not improve from 0.12946
1500/1500 [==============================] - 7s 5ms/step - loss: 0.0756 -
accuracy: 0.9867 - val_loss: 0.2186 - val_accuracy: 0.9778
Epoch 14/100
1491/1500 [============================>.] - ETA: 0s - loss: 0.0812 - accuracy:
0.9865
Epoch 00014: val_loss did not improve from 0.12946
1500/1500 [==============================] - 6s 4ms/step - loss: 0.0813 -
accuracy: 0.9865 - val_loss: 0.2346 - val_accuracy: 0.9778
Epoch 15/100
1491/1500 [============================>.] - ETA: 0s - loss: 0.0773 - accuracy:
0.9865
Epoch 00015: val_loss did not improve from 0.12946
1500/1500 [==============================] - 6s 4ms/step - loss: 0.0783 -
accuracy: 0.9865 - val_loss: 0.2618 - val_accuracy: 0.9772
Epoch 16/100
1490/1500 [============================>.] - ETA: 0s - loss: 0.0820 - accuracy:
0.9879
Epoch 00016: val_loss did not improve from 0.12946
1500/1500 [==============================] - 6s 4ms/step - loss: 0.0821 -
accuracy: 0.9880 - val_loss: 0.2316 - val_accuracy: 0.9787
Epoch 17/100
1496/1500 [============================>.] - ETA: 0s - loss: 0.0742 - accuracy:
0.9884
```

```
Epoch 00017: val_loss did not improve from 0.12946
1500/1500 [==============================] - 6s 4ms/step - loss: 0.0742 -
accuracy: 0.9884 - val_loss: 0.2217 - val_accuracy: 0.9790
Epoch 18/100
1496/1500 [============================>.] - ETA: 0s - loss: 0.0813 - accuracy:
0.9881
Epoch 00018: val_loss did not improve from 0.12946
1500/1500 [==============================] - 6s 4ms/step - loss: 0.0814 -
accuracy: 0.9881 - val_loss: 0.2616 - val_accuracy: 0.9774
Epoch 19/100
1492/1500 [============================>.] - ETA: 0s - loss: 0.0660 - accuracy:
0.9892
Epoch 00019: val_loss did not improve from 0.12946
1500/1500 [==============================] - 6s 4ms/step - loss: 0.0665 -
accuracy: 0.9891 - val_loss: 0.3383 - val_accuracy: 0.9768
Epoch 20/100
1500/1500 [==============================] - ETA: 0s - loss: 0.0710 - accuracy:
0.9897
Epoch 00020: val_loss did not improve from 0.12946
1500/1500 [==============================] - 6s 4ms/step - loss: 0.0710 -
accuracy: 0.9897 - val_loss: 0.3035 - val_accuracy: 0.9790
Epoch 21/100
1497/1500 [============================>.] - ETA: 0s - loss: 0.0621 - accuracy:
0.9900
Epoch 00021: val_loss did not improve from 0.12946
1500/1500 [==============================] - 6s 4ms/step - loss: 0.0620 -
accuracy: 0.9900 - val_loss: 0.3206 - val_accuracy: 0.9803
Epoch 22/100
1498/1500 [============================>.] - ETA: 0s - loss: 0.0718 - accuracy:
0.9906
Epoch 00022: val_loss did not improve from 0.12946
1500/1500 [==============================] - 6s 4ms/step - loss: 0.0717 -
accuracy: 0.9906 - val_loss: 0.3257 - val_accuracy: 0.9794
```

1.5.7 Plot CV Results

[20]:
```
pd.DataFrame(ffnn_history.history)[['accuracy', 'val_accuracy']].
 ↪plot(figsize=(14,4))
sns.despine();
```



1.5.8 Load the Best Model

[21]:
```
# load the weights that yielded the best validation accuracy
ffnn.load_weights(ffn_path.as_posix())
```

1.5.9 Test Classification Accuracy

[22]:
```
# evaluate test accuracy
ffnn_accuracy = ffnn.evaluate(X_test, y_test, verbose=0)[1]

print(f'Test accuracy: {ffnn_accuracy:.2%}')
```

```
Test accuracy: 97.29%
```

1.6 LeNet5

[23]:
```
K.clear_session()
```

1.6.1 Model Architecture

We can define a simplified version of LeNet5 that omits the original final layer containing radial basis functions as follows, using the default 'valid' padding and single step strides unless defined otherwise:

[24]:
```
lenet5 = Sequential([
    Conv2D(filters=6,
           kernel_size=5,
           activation='relu',
           input_shape=(28, 28, 1),
           name='CONV1'),
    AveragePooling2D(pool_size=(2, 2),
                     strides=(1, 1),
                     padding='valid',
                     name='POOL1'),
    Conv2D(filters=16, kernel_size=(5, 5), activation='tanh', name='CONV2'),
```

```
    AveragePooling2D(pool_size=(2, 2), strides=(2, 2), name='POOL2'),
    Conv2D(filters=120, kernel_size=(5, 5), activation='tanh', name='CONV3'),
    Flatten(name='FLAT'),
    Dense(units=84, activation='tanh', name='FC6'),
    Dense(units=10, activation='softmax', name='FC7')
])
```

The summary indicates that the model thus defined has over 300,000 parameters:

[25]:
```
lenet5.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
CONV1 (Conv2D)               (None, 24, 24, 6)         156
_________________________________________________________________
POOL1 (AveragePooling2D)     (None, 23, 23, 6)         0
_________________________________________________________________
CONV2 (Conv2D)               (None, 19, 19, 16)        2416
_________________________________________________________________
POOL2 (AveragePooling2D)     (None, 9, 9, 16)          0
_________________________________________________________________
CONV3 (Conv2D)               (None, 5, 5, 120)         48120
_________________________________________________________________
FLAT (Flatten)               (None, 3000)              0
_________________________________________________________________
FC6 (Dense)                  (None, 84)                252084
_________________________________________________________________
FC7 (Dense)                  (None, 10)                850
=================================================================
Total params: 303,626
Trainable params: 303,626
Non-trainable params: 0
_________________________________________________________________
```

We compile using crossentropy loss and the original stochastic gradient optimizer:

[26]:
```
lenet5.compile(loss='sparse_categorical_crossentropy',
               optimizer='SGD',
               metrics=['accuracy'])
```

1.6.2 Define checkpoint callback

[27]:
```
lenet_path = mnist_path / 'lenet.best.hdf5'
```

[28]:
```
checkpointer = ModelCheckpoint(filepath=lenet_path.as_posix(),
                               verbose=1,
                               save_best_only=True)
```

Now we are ready to train the model. The model expects 4D input so we reshape accordingly. We use the standard batch size of 32, 80-20 train-validation split, use checkpointing to store the model weights if the validation error improves, and make sure the dataset is randomly shuffled:

1.6.3 Train Model

[29]:
```
batch_size=32
epochs=100
```

[30]:
```
lenet_history = lenet5.fit(
    X_train.reshape(-1, 28, 28, 1),
    y_train,
    batch_size=batch_size,
    epochs=epochs,
    validation_split=0.2,  # use 0 to train on all data
    callbacks=[checkpointer, early_stopping],
    verbose=1,
    shuffle=True)
```

```
Epoch 1/100
1500/1500 [==============================] - ETA: 0s - loss: 0.4736 - accuracy:
0.8673
Epoch 00001: val_loss improved from inf to 0.23828, saving model to
results/mnist/lenet.best.hdf5
1500/1500 [==============================] - 12s 8ms/step - loss: 0.4736 -
accuracy: 0.8673 - val_loss: 0.2383 - val_accuracy: 0.9317
Epoch 2/100
1499/1500 [============================>.] - ETA: 0s - loss: 0.2040 - accuracy:
0.9416
Epoch 00002: val_loss improved from 0.23828 to 0.16032, saving model to
results/mnist/lenet.best.hdf5
1500/1500 [==============================] - 12s 8ms/step - loss: 0.2041 -
accuracy: 0.9416 - val_loss: 0.1603 - val_accuracy: 0.9571
Epoch 3/100
1500/1500 [==============================] - ETA: 0s - loss: 0.1415 - accuracy:
0.9598
Epoch 00003: val_loss improved from 0.16032 to 0.12653, saving model to
results/mnist/lenet.best.hdf5
1500/1500 [==============================] - 12s 8ms/step - loss: 0.1415 -
accuracy: 0.9598 - val_loss: 0.1265 - val_accuracy: 0.9634
Epoch 4/100
1499/1500 [============================>.] - ETA: 0s - loss: 0.1078 - accuracy:
0.9690
Epoch 00004: val_loss improved from 0.12653 to 0.09780, saving model to
```

```
results/mnist/lenet.best.hdf5
1500/1500 [==============================] - 12s 8ms/step - loss: 0.1078 -
accuracy: 0.9690 - val_loss: 0.0978 - val_accuracy: 0.9721
Epoch 5/100
1494/1500 [============================>.] - ETA: 0s - loss: 0.0882 - accuracy:
0.9745
Epoch 00005: val_loss improved from 0.09780 to 0.08991, saving model to
results/mnist/lenet.best.hdf5
1500/1500 [==============================] - 12s 8ms/step - loss: 0.0883 -
accuracy: 0.9745 - val_loss: 0.0899 - val_accuracy: 0.9732
Epoch 6/100
1498/1500 [============================>.] - ETA: 0s - loss: 0.0749 - accuracy:
0.9782
Epoch 00006: val_loss improved from 0.08991 to 0.07613, saving model to
results/mnist/lenet.best.hdf5
1500/1500 [==============================] - 12s 8ms/step - loss: 0.0748 -
accuracy: 0.9782 - val_loss: 0.0761 - val_accuracy: 0.9780
Epoch 7/100
1497/1500 [============================>.] - ETA: 0s - loss: 0.0659 - accuracy:
0.9808
Epoch 00007: val_loss improved from 0.07613 to 0.07121, saving model to
results/mnist/lenet.best.hdf5
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0659 -
accuracy: 0.9808 - val_loss: 0.0712 - val_accuracy: 0.9797
Epoch 8/100
1493/1500 [============================>.] - ETA: 0s - loss: 0.0590 - accuracy:
0.9830
Epoch 00008: val_loss improved from 0.07121 to 0.06667, saving model to
results/mnist/lenet.best.hdf5
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0589 -
accuracy: 0.9830 - val_loss: 0.0667 - val_accuracy: 0.9808
Epoch 9/100
1497/1500 [============================>.] - ETA: 0s - loss: 0.0529 - accuracy:
0.9850
Epoch 00009: val_loss improved from 0.06667 to 0.06278, saving model to
results/mnist/lenet.best.hdf5
1500/1500 [==============================] - 11s 7ms/step - loss: 0.0529 -
accuracy: 0.9850 - val_loss: 0.0628 - val_accuracy: 0.9812
Epoch 10/100
1497/1500 [============================>.] - ETA: 0s - loss: 0.0485 - accuracy:
0.9861
Epoch 00010: val_loss improved from 0.06278 to 0.05955, saving model to
results/mnist/lenet.best.hdf5
1500/1500 [==============================] - 12s 8ms/step - loss: 0.0484 -
accuracy: 0.9861 - val_loss: 0.0595 - val_accuracy: 0.9816
Epoch 11/100
1494/1500 [============================>.] - ETA: 0s - loss: 0.0446 - accuracy:
0.9868
```

```
Epoch 00011: val_loss improved from 0.05955 to 0.05720, saving model to
results/mnist/lenet.best.hdf5
1500/1500 [==============================] - 11s 7ms/step - loss: 0.0445 -
accuracy: 0.9868 - val_loss: 0.0572 - val_accuracy: 0.9827
Epoch 12/100
1495/1500 [============================>.] - ETA: 0s - loss: 0.0415 - accuracy:
0.9883
Epoch 00012: val_loss improved from 0.05720 to 0.05526, saving model to
results/mnist/lenet.best.hdf5
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0415 -
accuracy: 0.9883 - val_loss: 0.0553 - val_accuracy: 0.9833
Epoch 13/100
1496/1500 [============================>.] - ETA: 0s - loss: 0.0383 - accuracy:
0.9896
Epoch 00013: val_loss improved from 0.05526 to 0.05236, saving model to
results/mnist/lenet.best.hdf5
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0382 -
accuracy: 0.9896 - val_loss: 0.0524 - val_accuracy: 0.9846
Epoch 14/100
1496/1500 [============================>.] - ETA: 0s - loss: 0.0359 - accuracy:
0.9897
Epoch 00014: val_loss improved from 0.05236 to 0.05134, saving model to
results/mnist/lenet.best.hdf5
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0359 -
accuracy: 0.9897 - val_loss: 0.0513 - val_accuracy: 0.9844
Epoch 15/100
1498/1500 [============================>.] - ETA: 0s - loss: 0.0335 - accuracy:
0.9904
Epoch 00015: val_loss improved from 0.05134 to 0.04950, saving model to
results/mnist/lenet.best.hdf5
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0335 -
accuracy: 0.9904 - val_loss: 0.0495 - val_accuracy: 0.9852
Epoch 16/100
1493/1500 [============================>.] - ETA: 0s - loss: 0.0315 - accuracy:
0.9914
Epoch 00016: val_loss did not improve from 0.04950
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0315 -
accuracy: 0.9914 - val_loss: 0.0499 - val_accuracy: 0.9847
Epoch 17/100
1499/1500 [============================>.] - ETA: 0s - loss: 0.0296 - accuracy:
0.9918
Epoch 00017: val_loss improved from 0.04950 to 0.04914, saving model to
results/mnist/lenet.best.hdf5
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0296 -
accuracy: 0.9918 - val_loss: 0.0491 - val_accuracy: 0.9844
Epoch 18/100
1499/1500 [============================>.] - ETA: 0s - loss: 0.0280 - accuracy:
0.9924
```

```
Epoch 00018: val_loss improved from 0.04914 to 0.04556, saving model to
results/mnist/lenet.best.hdf5
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0280 -
accuracy: 0.9924 - val_loss: 0.0456 - val_accuracy: 0.9871
Epoch 19/100
1492/1500 [============================>.] - ETA: 0s - loss: 0.0261 - accuracy:
0.9931
Epoch 00019: val_loss did not improve from 0.04556
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0262 -
accuracy: 0.9931 - val_loss: 0.0458 - val_accuracy: 0.9862
Epoch 20/100
1493/1500 [============================>.] - ETA: 0s - loss: 0.0248 - accuracy:
0.9934
Epoch 00020: val_loss improved from 0.04556 to 0.04541, saving model to
results/mnist/lenet.best.hdf5
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0248 -
accuracy: 0.9934 - val_loss: 0.0454 - val_accuracy: 0.9862
Epoch 21/100
1500/1500 [==============================] - ETA: 0s - loss: 0.0234 - accuracy:
0.9938
Epoch 00021: val_loss did not improve from 0.04541
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0234 -
accuracy: 0.9938 - val_loss: 0.0475 - val_accuracy: 0.9850
Epoch 22/100
1496/1500 [============================>.] - ETA: 0s - loss: 0.0223 - accuracy:
0.9941
Epoch 00022: val_loss improved from 0.04541 to 0.04413, saving model to
results/mnist/lenet.best.hdf5
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0222 -
accuracy: 0.9941 - val_loss: 0.0441 - val_accuracy: 0.9868
Epoch 23/100
1493/1500 [============================>.] - ETA: 0s - loss: 0.0210 - accuracy:
0.9949
Epoch 00023: val_loss improved from 0.04413 to 0.04322, saving model to
results/mnist/lenet.best.hdf5
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0209 -
accuracy: 0.9949 - val_loss: 0.0432 - val_accuracy: 0.9875
Epoch 24/100
1498/1500 [============================>.] - ETA: 0s - loss: 0.0202 - accuracy:
0.9950
Epoch 00024: val_loss improved from 0.04322 to 0.04182, saving model to
results/mnist/lenet.best.hdf5
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0202 -
accuracy: 0.9950 - val_loss: 0.0418 - val_accuracy: 0.9877
Epoch 25/100
1497/1500 [============================>.] - ETA: 0s - loss: 0.0187 - accuracy:
0.9959
Epoch 00025: val_loss improved from 0.04182 to 0.04179, saving model to
```

```
results/mnist/lenet.best.hdf5
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0187 -
accuracy: 0.9959 - val_loss: 0.0418 - val_accuracy: 0.9879
Epoch 26/100
1492/1500 [============================>.] - ETA: 0s - loss: 0.0182 - accuracy:
0.9955
Epoch 00026: val_loss did not improve from 0.04179
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0181 -
accuracy: 0.9955 - val_loss: 0.0449 - val_accuracy: 0.9858
Epoch 27/100
1495/1500 [============================>.] - ETA: 0s - loss: 0.0174 - accuracy:
0.9958
Epoch 00027: val_loss did not improve from 0.04179
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0174 -
accuracy: 0.9958 - val_loss: 0.0418 - val_accuracy: 0.9872
Epoch 28/100
1500/1500 [==============================] - ETA: 0s - loss: 0.0164 - accuracy:
0.9962
Epoch 00028: val_loss did not improve from 0.04179
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0164 -
accuracy: 0.9962 - val_loss: 0.0425 - val_accuracy: 0.9870
Epoch 29/100
1499/1500 [============================>.] - ETA: 0s - loss: 0.0155 - accuracy:
0.9967
Epoch 00029: val_loss did not improve from 0.04179
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0155 -
accuracy: 0.9967 - val_loss: 0.0425 - val_accuracy: 0.9877
Epoch 30/100
1497/1500 [============================>.] - ETA: 0s - loss: 0.0150 - accuracy:
0.9967
Epoch 00030: val_loss improved from 0.04179 to 0.04099, saving model to
results/mnist/lenet.best.hdf5
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0150 -
accuracy: 0.9967 - val_loss: 0.0410 - val_accuracy: 0.9874
Epoch 31/100
1497/1500 [============================>.] - ETA: 0s - loss: 0.0142 - accuracy:
0.9972
Epoch 00031: val_loss improved from 0.04099 to 0.04070, saving model to
results/mnist/lenet.best.hdf5
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0142 -
accuracy: 0.9972 - val_loss: 0.0407 - val_accuracy: 0.9873
Epoch 32/100
1493/1500 [============================>.] - ETA: 0s - loss: 0.0136 - accuracy:
0.9974
Epoch 00032: val_loss did not improve from 0.04070
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0135 -
accuracy: 0.9974 - val_loss: 0.0412 - val_accuracy: 0.9877
Epoch 33/100
```

```
1497/1500 [============================>.] - ETA: 0s - loss: 0.0129 - accuracy:
0.9974
Epoch 00033: val_loss improved from 0.04070 to 0.03972, saving model to
results/mnist/lenet.best.hdf5
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0129 -
accuracy: 0.9974 - val_loss: 0.0397 - val_accuracy: 0.9877
Epoch 34/100
1496/1500 [============================>.] - ETA: 0s - loss: 0.0123 - accuracy:
0.9976
Epoch 00034: val_loss did not improve from 0.03972
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0123 -
accuracy: 0.9976 - val_loss: 0.0398 - val_accuracy: 0.9881
Epoch 35/100
1500/1500 [==============================] - ETA: 0s - loss: 0.0118 - accuracy:
0.9979
Epoch 00035: val_loss did not improve from 0.03972
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0118 -
accuracy: 0.9979 - val_loss: 0.0400 - val_accuracy: 0.9882
Epoch 36/100
1500/1500 [==============================] - ETA: 0s - loss: 0.0113 - accuracy:
0.9980
Epoch 00036: val_loss did not improve from 0.03972
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0113 -
accuracy: 0.9980 - val_loss: 0.0401 - val_accuracy: 0.9874
Epoch 37/100
1494/1500 [============================>.] - ETA: 0s - loss: 0.0109 - accuracy:
0.9981
Epoch 00037: val_loss improved from 0.03972 to 0.03957, saving model to
results/mnist/lenet.best.hdf5
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0109 -
accuracy: 0.9981 - val_loss: 0.0396 - val_accuracy: 0.9883
Epoch 38/100
1492/1500 [============================>.] - ETA: 0s - loss: 0.0104 - accuracy:
0.9983
Epoch 00038: val_loss improved from 0.03957 to 0.03874, saving model to
results/mnist/lenet.best.hdf5
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0104 -
accuracy: 0.9983 - val_loss: 0.0387 - val_accuracy: 0.9882
Epoch 39/100
1494/1500 [============================>.] - ETA: 0s - loss: 0.0101 - accuracy:
0.9983
Epoch 00039: val_loss did not improve from 0.03874
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0100 -
accuracy: 0.9983 - val_loss: 0.0399 - val_accuracy: 0.9883
Epoch 40/100
1495/1500 [============================>.] - ETA: 0s - loss: 0.0096 - accuracy:
0.9986
Epoch 00040: val_loss did not improve from 0.03874
```

```
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0096 -
accuracy: 0.9986 - val_loss: 0.0398 - val_accuracy: 0.9885
Epoch 41/100
1496/1500 [============================>.] - ETA: 0s - loss: 0.0091 - accuracy:
0.9987
Epoch 00041: val_loss did not improve from 0.03874
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0092 -
accuracy: 0.9987 - val_loss: 0.0389 - val_accuracy: 0.9882
Epoch 42/100
1493/1500 [============================>.] - ETA: 0s - loss: 0.0087 - accuracy:
0.9988
Epoch 00042: val_loss did not improve from 0.03874
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0088 -
accuracy: 0.9987 - val_loss: 0.0390 - val_accuracy: 0.9885
Epoch 43/100
1499/1500 [============================>.] - ETA: 0s - loss: 0.0084 - accuracy:
0.9990
Epoch 00043: val_loss did not improve from 0.03874
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0084 -
accuracy: 0.9990 - val_loss: 0.0405 - val_accuracy: 0.9882
Epoch 44/100
1499/1500 [============================>.] - ETA: 0s - loss: 0.0082 - accuracy:
0.9988
Epoch 00044: val_loss did not improve from 0.03874
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0082 -
accuracy: 0.9988 - val_loss: 0.0391 - val_accuracy: 0.9879
Epoch 45/100
1499/1500 [============================>.] - ETA: 0s - loss: 0.0078 - accuracy:
0.9990
Epoch 00045: val_loss improved from 0.03874 to 0.03856, saving model to
results/mnist/lenet.best.hdf5
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0078 -
accuracy: 0.9990 - val_loss: 0.0386 - val_accuracy: 0.9892
Epoch 46/100
1499/1500 [============================>.] - ETA: 0s - loss: 0.0076 - accuracy:
0.9990
Epoch 00046: val_loss improved from 0.03856 to 0.03762, saving model to
results/mnist/lenet.best.hdf5
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0076 -
accuracy: 0.9990 - val_loss: 0.0376 - val_accuracy: 0.9897
Epoch 47/100
1496/1500 [============================>.] - ETA: 0s - loss: 0.0073 - accuracy:
0.9990
Epoch 00047: val_loss did not improve from 0.03762
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0073 -
accuracy: 0.9990 - val_loss: 0.0387 - val_accuracy: 0.9891
Epoch 48/100
1498/1500 [============================>.] - ETA: 0s - loss: 0.0070 - accuracy:
```

```
0.9990
Epoch 00048: val_loss did not improve from 0.03762
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0070 -
accuracy: 0.9990 - val_loss: 0.0399 - val_accuracy: 0.9880
Epoch 49/100
1498/1500 [============================>.] - ETA: 0s - loss: 0.0068 - accuracy:
0.9991
Epoch 00049: val_loss did not improve from 0.03762
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0068 -
accuracy: 0.9991 - val_loss: 0.0377 - val_accuracy: 0.9889
Epoch 50/100
1496/1500 [============================>.] - ETA: 0s - loss: 0.0066 - accuracy:
0.9992
Epoch 00050: val_loss did not improve from 0.03762
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0066 -
accuracy: 0.9992 - val_loss: 0.0400 - val_accuracy: 0.9875
Epoch 51/100
1500/1500 [==============================] - ETA: 0s - loss: 0.0063 - accuracy:
0.9993
Epoch 00051: val_loss did not improve from 0.03762
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0063 -
accuracy: 0.9993 - val_loss: 0.0406 - val_accuracy: 0.9881
Epoch 52/100
1498/1500 [============================>.] - ETA: 0s - loss: 0.0061 - accuracy:
0.9992
Epoch 00052: val_loss did not improve from 0.03762
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0061 -
accuracy: 0.9992 - val_loss: 0.0382 - val_accuracy: 0.9887
Epoch 53/100
1498/1500 [============================>.] - ETA: 0s - loss: 0.0060 - accuracy:
0.9993
Epoch 00053: val_loss did not improve from 0.03762
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0060 -
accuracy: 0.9993 - val_loss: 0.0387 - val_accuracy: 0.9885
Epoch 54/100
1495/1500 [============================>.] - ETA: 0s - loss: 0.0057 - accuracy:
0.9994
Epoch 00054: val_loss did not improve from 0.03762
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0057 -
accuracy: 0.9994 - val_loss: 0.0388 - val_accuracy: 0.9877
Epoch 55/100
1500/1500 [==============================] - ETA: 0s - loss: 0.0055 - accuracy:
0.9994
Epoch 00055: val_loss did not improve from 0.03762
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0055 -
accuracy: 0.9994 - val_loss: 0.0381 - val_accuracy: 0.9892
Epoch 56/100
1498/1500 [============================>.] - ETA: 0s - loss: 0.0054 - accuracy:
```

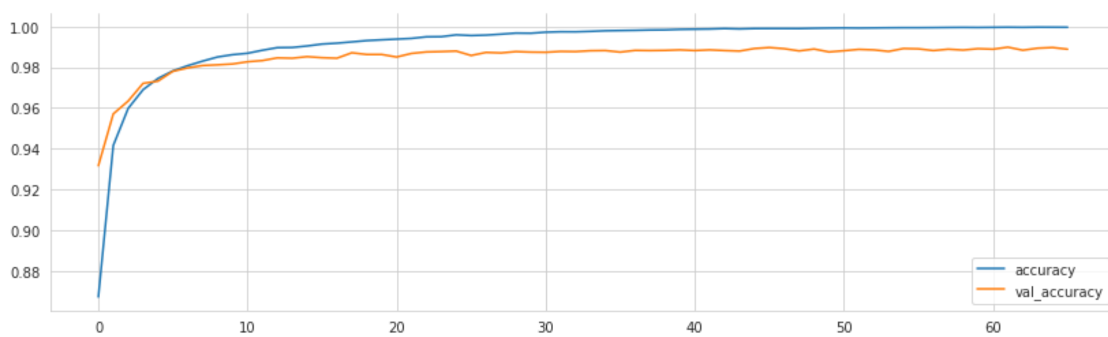
```
0.9994
Epoch 00056: val_loss did not improve from 0.03762
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0054 -
accuracy: 0.9994 - val_loss: 0.0391 - val_accuracy: 0.9890
Epoch 57/100
1500/1500 [==============================] - ETA: 0s - loss: 0.0052 - accuracy:
0.9994
Epoch 00057: val_loss did not improve from 0.03762
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0052 -
accuracy: 0.9994 - val_loss: 0.0384 - val_accuracy: 0.9882
Epoch 58/100
1492/1500 [============================>.] - ETA: 0s - loss: 0.0050 - accuracy:
0.9995
Epoch 00058: val_loss did not improve from 0.03762
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0051 -
accuracy: 0.9995 - val_loss: 0.0384 - val_accuracy: 0.9888
Epoch 59/100
1495/1500 [============================>.] - ETA: 0s - loss: 0.0048 - accuracy:
0.9995
Epoch 00059: val_loss did not improve from 0.03762
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0048 -
accuracy: 0.9995 - val_loss: 0.0380 - val_accuracy: 0.9884
Epoch 60/100
1495/1500 [============================>.] - ETA: 0s - loss: 0.0047 - accuracy:
0.9995
Epoch 00060: val_loss did not improve from 0.03762
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0047 -
accuracy: 0.9995 - val_loss: 0.0385 - val_accuracy: 0.9891
Epoch 61/100
1499/1500 [============================>.] - ETA: 0s - loss: 0.0046 - accuracy:
0.9996
Epoch 00061: val_loss did not improve from 0.03762
1500/1500 [==============================] - 11s 7ms/step - loss: 0.0046 -
accuracy: 0.9996 - val_loss: 0.0381 - val_accuracy: 0.9888
Epoch 62/100
1494/1500 [============================>.] - ETA: 0s - loss: 0.0044 - accuracy:
0.9996
Epoch 00062: val_loss did not improve from 0.03762
1500/1500 [==============================] - 11s 7ms/step - loss: 0.0044 -
accuracy: 0.9996 - val_loss: 0.0379 - val_accuracy: 0.9898
Epoch 63/100
1495/1500 [============================>.] - ETA: 0s - loss: 0.0044 - accuracy:
0.9996
Epoch 00063: val_loss did not improve from 0.03762
1500/1500 [==============================] - 12s 8ms/step - loss: 0.0044 -
accuracy: 0.9996 - val_loss: 0.0386 - val_accuracy: 0.9883
Epoch 64/100
1499/1500 [============================>.] - ETA: 0s - loss: 0.0042 - accuracy:
```

```
0.9996
Epoch 00064: val_loss did not improve from 0.03762
1500/1500 [==============================] - 11s 7ms/step - loss: 0.0042 -
accuracy: 0.9996 - val_loss: 0.0383 - val_accuracy: 0.9893
Epoch 65/100
1495/1500 [============================>.] - ETA: 0s - loss: 0.0041 - accuracy:
0.9996
Epoch 00065: val_loss did not improve from 0.03762
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0041 -
accuracy: 0.9996 - val_loss: 0.0380 - val_accuracy: 0.9897
Epoch 66/100
1497/1500 [============================>.] - ETA: 0s - loss: 0.0040 - accuracy:
0.9996
Epoch 00066: val_loss did not improve from 0.03762
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0040 -
accuracy: 0.9996 - val_loss: 0.0385 - val_accuracy: 0.9888
```

1.6.4 Plot CV Results

On a single GPU, 50 epochs take around 2.5 minutes, resulting in a test accuracy of 99.09%, slightly below the same result as for the original LeNet5:

[31]:
```
pd.DataFrame(lenet_history.history)[['accuracy', 'val_accuracy']].
 ↪plot(figsize=(14,4))
sns.despine();
```



1.6.5 Test Classification Accuracy

[32]:
```
# evaluate test accuracy
lenet_accuracy = lenet5.evaluate(X_test.reshape(-1, 28, 28, 1), y_test,␣
 ↪verbose=0)[1]
print('Test accuracy: {:.2%}'.format(lenet_accuracy))
```

```
Test accuracy: 98.97%
```

1.7 Summary

For comparison, a simple two-layer feedforward network achieves only 37.36% test accuracy.

The LeNet5 improvement on MNIST is, in fact, modest. Non-neural methods have also achieved classification accuracies greater than or equal to 99%, including K-Nearest Neighbours or Support Vector Machines. CNNs really shine with more challenging datasets as we will see next.