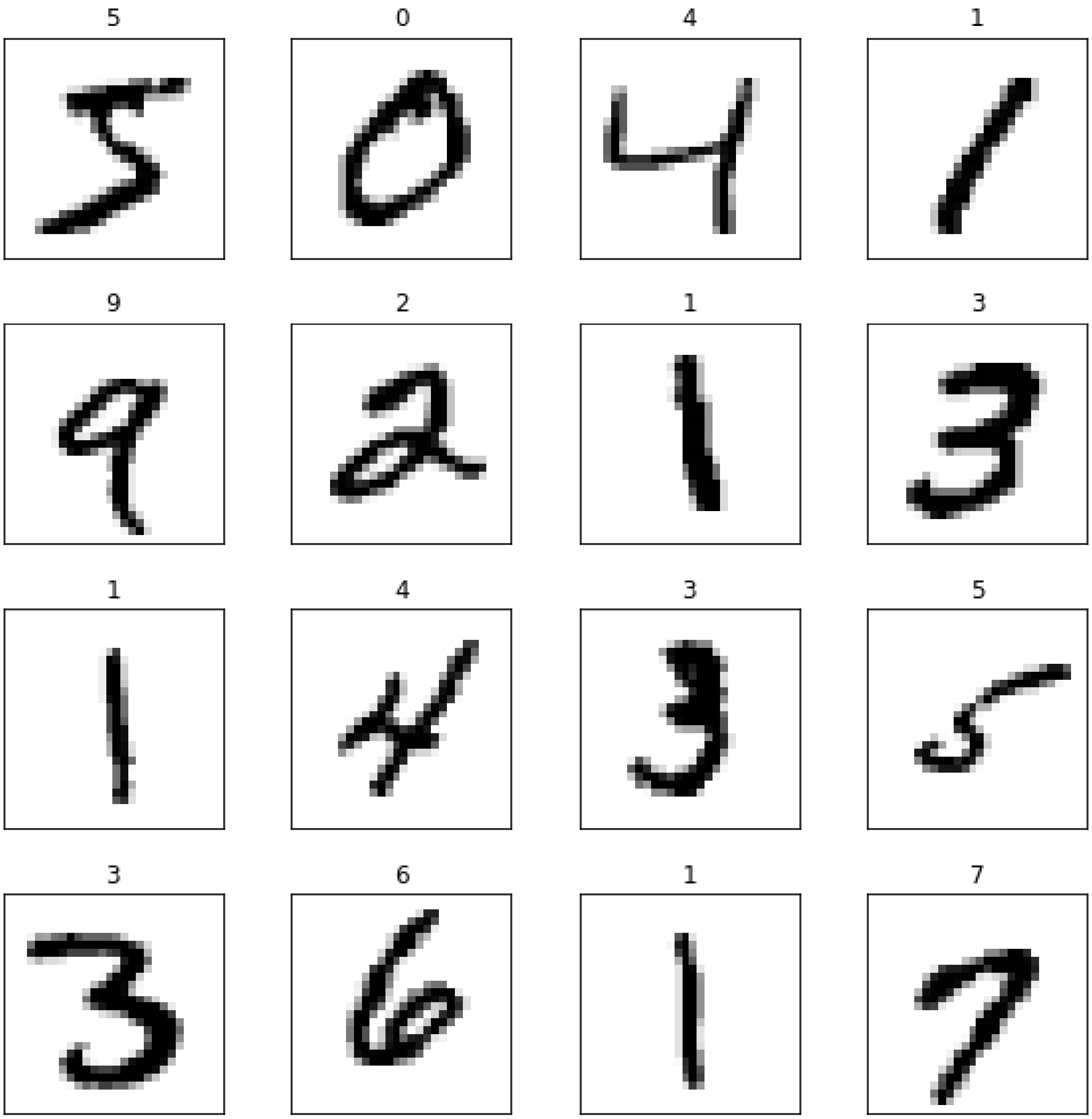
Load MNIST digits

```python
# copied from professor Dirk's code

import tensorflow as tf
from keras.datasets import mnist
(train_X, train_y), (test_X, test_y) = mnist.load_data()
epochs = 10
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
```

Data analysis

```python
# copied from Dirk's example

import matplotlib.pyplot as plt

train_images = train_X / 255.0
test_images = test_X / 255.0

plt.figure(figsize=(10, 10))

for i in range(16):
    plt.subplot(4, 4, i + 1)
    plt.subplots_adjust(hspace=.3)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.title(train_y[i])
plt.show()
```




```
# copied from Dirk's example

X_train = train_images.reshape((train_images.shape[0], 28, 28, 1))
X_test = test_images.reshape((test_images.shape[0], 28, 28, 1))

print(X_train.shape)

tf.random.set_seed(42)
```

```
(60000, 28, 28, 1)
```

Shallow network (no augmentation)

```
# based on Dirk's example

from tensorflow.keras import datasets, layers, models, losses
```

```
# made it into a function so I can create more models later just by
calling
def get_shallow():
  model_shallow = models.Sequential()
  model_shallow.add(layers.Conv2D(32, (3, 3), activation='relu',
input_shape=(28, 28, 1)))
  model_shallow.add(layers.MaxPooling2D((2, 2)))
  model_shallow.add(layers.Conv2D(64, (3, 3), activation='relu'))
  model_shallow.add(layers.MaxPooling2D((2, 2)))
  model_shallow.add(layers.Conv2D(128, (3, 3), activation='relu'))
  model_shallow.add(layers.Flatten())
  model_shallow.add(layers.Dense(64, activation='relu'))
  model_shallow.add(layers.Dense(10, activation='softmax'))

  model_shallow.compile(optimizer='adam',
                loss=losses.sparse_categorical_crossentropy,
                metrics=['accuracy'])
  return model_shallow

model_shallow = get_shallow()
model_shallow.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320       
                                                                 
 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0         
 )                                                               
                                                                 
 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496     
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0         
 2D)                                                             
                                                                 
 conv2d_2 (Conv2D)           (None, 3, 3, 128)         73856     
                                                                 
 flatten (Flatten)           (None, 1152)              0         
                                                                 
 dense (Dense)               (None, 64)                73792     
                                                                 
 dense_1 (Dense)             (None, 10)                650       
                                                                 
=================================================================
Total params: 167,114
Trainable params: 167,114
Non-trainable params: 0
_________________________________________________________________
```

Testing accuracy

```
# the test accuracy for a shallow model over MNIST looks good.

model_shallow = get_shallow()
model_shallow.fit(X_train, train_y, validation_data=(X_test, test_y),
epochs=epochs)

test_loss, test_acc = model_shallow.evaluate(X_test, test_y,
verbose=2)

print('Accuracy on test set:', test_acc)

predictions = model_shallow.predict(X_test)
print('Predictions for the first image:')
print(predictions[0])
```

```
Epoch 1/10
1875/1875 [==============================] - 17s 9ms/step - loss:
0.1284 - accuracy: 0.9608 - val_loss: 0.0373 - val_accuracy: 0.9871
Epoch 2/10
1875/1875 [==============================] - 16s 9ms/step - loss:
0.0432 - accuracy: 0.9864 - val_loss: 0.0592 - val_accuracy: 0.9798
Epoch 3/10
1875/1875 [==============================] - 16s 8ms/step - loss:
0.0310 - accuracy: 0.9903 - val_loss: 0.0244 - val_accuracy: 0.9930
Epoch 4/10
1875/1875 [==============================] - 16s 8ms/step - loss:
0.0228 - accuracy: 0.9927 - val_loss: 0.0249 - val_accuracy: 0.9932
Epoch 5/10
1875/1875 [==============================] - 16s 9ms/step - loss:
0.0170 - accuracy: 0.9946 - val_loss: 0.0287 - val_accuracy: 0.9912
Epoch 6/10
1875/1875 [==============================] - 15s 8ms/step - loss:
0.0152 - accuracy: 0.9953 - val_loss: 0.0324 - val_accuracy: 0.9904
Epoch 7/10
1875/1875 [==============================] - 15s 8ms/step - loss:
0.0124 - accuracy: 0.9961 - val_loss: 0.0293 - val_accuracy: 0.9918
Epoch 8/10
1875/1875 [==============================] - 15s 8ms/step - loss:
0.0111 - accuracy: 0.9963 - val_loss: 0.0394 - val_accuracy: 0.9888
Epoch 9/10
1875/1875 [==============================] - 15s 8ms/step - loss:
0.0090 - accuracy: 0.9974 - val_loss: 0.0351 - val_accuracy: 0.9901
Epoch 10/10
1875/1875 [==============================] - 15s 8ms/step - loss:
0.0082 - accuracy: 0.9976 - val_loss: 0.0301 - val_accuracy: 0.9915
313/313 - 1s - loss: 0.0301 - accuracy: 0.9915 - 1s/epoch - 3ms/step
Accuracy on test set: 0.9915000200271606
Predictions for the first image:
```

```
[4.5378323e-11 7.7408968e-09 2.9041724e-08 2.5175821e-09 3.7288701e-05
 7.6123023e-12 1.3058901e-13 9.9996185e-01 1.4703166e-10 8.5536880e-
07]
```

Deep Network (no augmentation)

```
# created a deep model with 9 layers
# I had to reduce the number of maxpooling layers so that the input
for the softmax layer wouldn't be too small
def get_deep():
  model_deep = models.Sequential()
  model_deep.add(layers.Conv2D(32, (2, 2), activation='relu',
input_shape=(28, 28, 1)))
  model_deep.add(layers.Conv2D(64, (2, 2), activation='relu'))
  model_deep.add(layers.MaxPooling2D((2, 2)))
  model_deep.add(layers.Conv2D(128, (2, 2), activation='relu'))
  model_deep.add(layers.Conv2D(128, (2, 2), activation='relu'))
  model_deep.add(layers.MaxPooling2D((2, 2)))
  model_deep.add(layers.Conv2D(256, (2, 2), activation='relu'))
  model_deep.add(layers.Conv2D(256, (2, 2), activation='relu'))
  model_deep.add(layers.Conv2D(256, (2, 2), activation='relu'))
  model_deep.add(layers.Flatten())
  model_deep.add(layers.Dense(64, activation='relu'))
  model_deep.add(layers.Dense(10, activation='softmax'))

  model_deep.compile(optimizer='adam',
              loss=losses.sparse_categorical_crossentropy,
              metrics=['accuracy'])
  return model_deep

model_deep = get_deep()
model_deep.summary()
```

```
Model: "sequential_1"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 27, 27, 32)        160

 conv2d_4 (Conv2D)           (None, 26, 26, 64)        8256

 max_pooling2d_2 (MaxPooling  (None, 13, 13, 64)       0
 2D)

 conv2d_5 (Conv2D)           (None, 12, 12, 128)       32896

 conv2d_6 (Conv2D)           (None, 11, 11, 128)       65664

 max_pooling2d_3 (MaxPooling  (None, 5, 5, 128)        0
 2D)
```

```
 conv2d_7 (Conv2D)           (None, 4, 4, 256)         131328

 conv2d_8 (Conv2D)           (None, 3, 3, 256)         262400

 conv2d_9 (Conv2D)           (None, 2, 2, 256)         262400

 flatten_1 (Flatten)         (None, 1024)              0

 dense_2 (Dense)             (None, 64)                65600

 dense_3 (Dense)             (None, 10)                650

=================================================================
Total params: 829,354
Trainable params: 829,354
Non-trainable params: 0
_________________________________________________________________
```

Testing accuracy

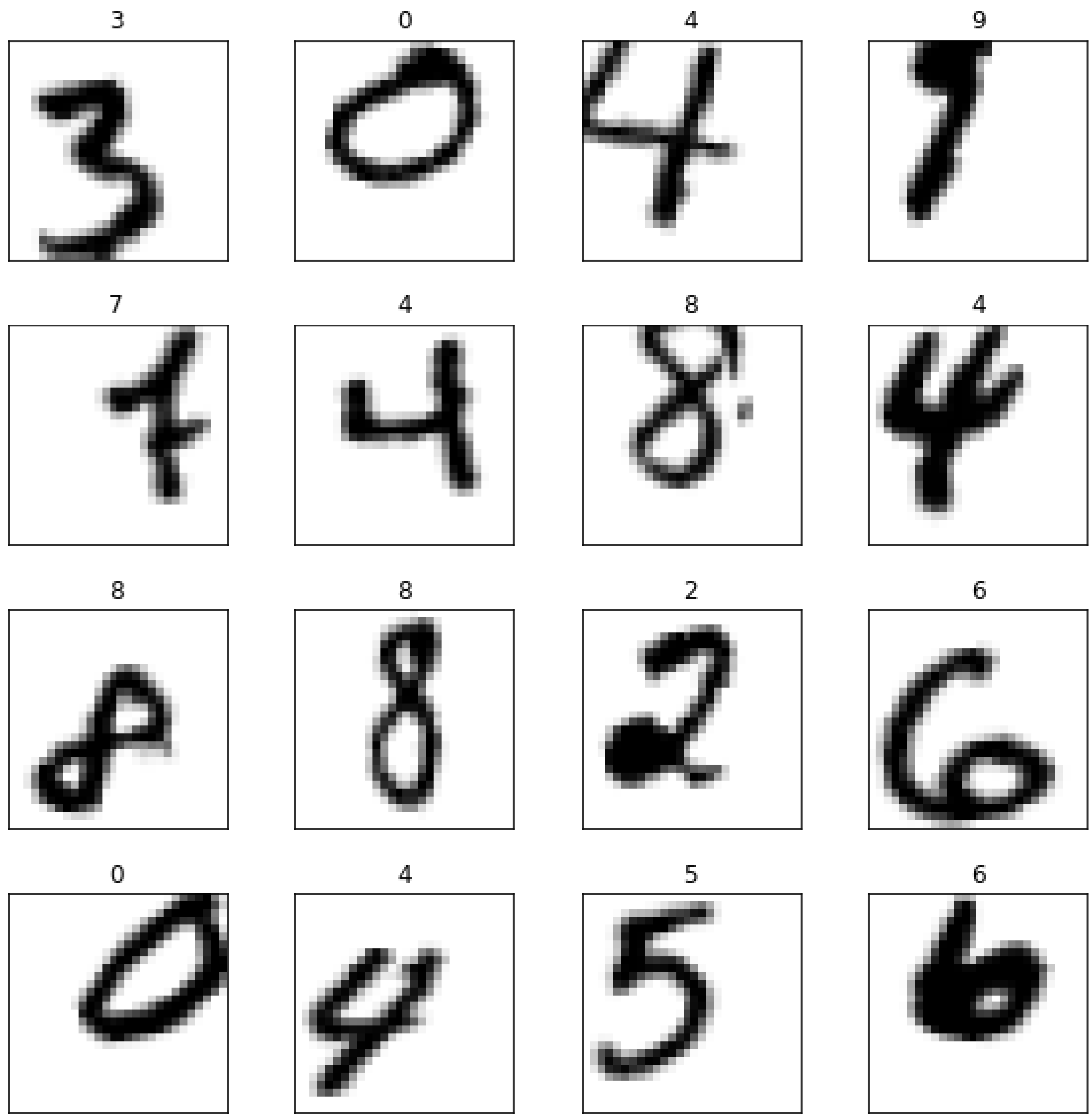
```
model_deep = get_deep()
model_deep.fit(X_train, train_y, validation_data=(X_test, test_y),
epochs=epochs)

test_loss, test_acc = model_deep.evaluate(X_test, test_y, verbose=2)

print('Accuracy on test set:', test_acc)

predictions = model_deep.predict(X_test)
print('Predictions for the first image:')
print(predictions[0])
```

```
Epoch 1/10
1875/1875 [==============================] - 25s 13ms/step - loss:
0.1441 - accuracy: 0.9546 - val_loss: 0.0580 - val_accuracy: 0.9821
Epoch 2/10
1875/1875 [==============================] - 25s 13ms/step - loss:
0.0526 - accuracy: 0.9842 - val_loss: 0.0599 - val_accuracy: 0.9818
Epoch 3/10
1875/1875 [==============================] - 24s 13ms/step - loss:
0.0376 - accuracy: 0.9884 - val_loss: 0.0332 - val_accuracy: 0.9908
Epoch 4/10
1875/1875 [==============================] - 25s 13ms/step - loss:
0.0302 - accuracy: 0.9910 - val_loss: 0.0330 - val_accuracy: 0.9898
Epoch 5/10
1875/1875 [==============================] - 24s 13ms/step - loss:
0.0262 - accuracy: 0.9924 - val_loss: 0.0406 - val_accuracy: 0.9887
Epoch 6/10
1875/1875 [==============================] - 24s 13ms/step - loss:
```

```
0.0223 - accuracy: 0.9933 - val_loss: 0.0471 - val_accuracy: 0.9877
Epoch 7/10
1875/1875 [==============================] - 24s 13ms/step - loss:
0.0202 - accuracy: 0.9939 - val_loss: 0.0405 - val_accuracy: 0.9905
Epoch 8/10
1875/1875 [==============================] - 24s 13ms/step - loss:
0.0187 - accuracy: 0.9947 - val_loss: 0.0418 - val_accuracy: 0.9886
Epoch 9/10
1875/1875 [==============================] - 24s 13ms/step - loss:
0.0166 - accuracy: 0.9953 - val_loss: 0.0484 - val_accuracy: 0.9888
Epoch 10/10
1875/1875 [==============================] - 25s 14ms/step - loss:
0.0160 - accuracy: 0.9952 - val_loss: 0.0372 - val_accuracy: 0.9920
313/313 - 2s - loss: 0.0372 - accuracy: 0.9920 - 2s/epoch - 5ms/step
Accuracy on test set: 0.9919999837875366
Predictions for the first image:
[1.2716915e-13 3.7804990e-09 3.6497346e-12 1.2005044e-10 7.3451599e-11
 2.6823122e-11 1.9595822e-19 1.0000000e+00 3.2460035e-13 3.9512715e-
10]
```

The test accuracy is slightly better than the shallow network.

```
## Data augmentation

# created the following augmentation that:
# rotates the digits, shift them vertically and horizontally, shear
and zoom them
augment = tf.keras.preprocessing.image.ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
)

augment.fit(X_train)

# demonstration of the augmentations
plt.figure(figsize=(10, 10))

X_batch, y_batch = next(augment.flow(X_train, train_y, batch_size=16))

for i in range(16):
    plt.subplot(4, 4, i + 1)
    plt.subplots_adjust(hspace=.3)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(X_batch[i, :, :, 0], cmap=plt.cm.binary)
    plt.title(y_batch[i])
plt.show()
```



Shallow network model (with data augmentation)

```
# shallow model performed slightly better on the test set with
augmentations
model_shallow = get_shallow()
model_shallow.fit(augment.flow(X_train, train_y),
validation_data=(X_test, test_y), epochs=epochs)

test_loss, test_acc = model_shallow.evaluate(X_test, test_y,
verbose=2)

print('Accuracy on test set:', test_acc)

predictions = model_shallow.predict(X_test)
print('Predictions for the first image:')
print(predictions[0])
```

```
Epoch 1/10
1875/1875 [==============================] - 35s 18ms/step - loss:
0.4996 - accuracy: 0.8396 - val_loss: 0.0892 - val_accuracy: 0.9722
Epoch 2/10
1875/1875 [==============================] - 34s 18ms/step - loss:
0.1834 - accuracy: 0.9430 - val_loss: 0.0494 - val_accuracy: 0.9842
Epoch 3/10
1875/1875 [==============================] - 36s 19ms/step - loss:
0.1321 - accuracy: 0.9587 - val_loss: 0.0464 - val_accuracy: 0.9841
Epoch 4/10
1875/1875 [==============================] - 38s 20ms/step - loss:
0.1097 - accuracy: 0.9664 - val_loss: 0.0387 - val_accuracy: 0.9881
Epoch 5/10
1875/1875 [==============================] - 38s 20ms/step - loss:
0.0971 - accuracy: 0.9709 - val_loss: 0.0425 - val_accuracy: 0.9866
Epoch 6/10
1875/1875 [==============================] - 38s 20ms/step - loss:
0.0890 - accuracy: 0.9730 - val_loss: 0.0463 - val_accuracy: 0.9856
Epoch 7/10
1875/1875 [==============================] - 38s 20ms/step - loss:
0.0833 - accuracy: 0.9746 - val_loss: 0.0314 - val_accuracy: 0.9899
Epoch 8/10
1875/1875 [==============================] - 37s 20ms/step - loss:
0.0741 - accuracy: 0.9774 - val_loss: 0.0306 - val_accuracy: 0.9914
Epoch 9/10
1875/1875 [==============================] - 38s 20ms/step - loss:
0.0702 - accuracy: 0.9784 - val_loss: 0.0393 - val_accuracy: 0.9890
Epoch 10/10
1875/1875 [==============================] - 37s 20ms/step - loss:
0.0661 - accuracy: 0.9803 - val_loss: 0.0267 - val_accuracy: 0.9919
313/313 - 1s - loss: 0.0267 - accuracy: 0.9919 - 1s/epoch - 3ms/step
Accuracy on test set: 0.9919000267982483
Predictions for the first image:
[7.91083399e-13 1.52823518e-10 1.34597747e-06 2.51730772e-11
 6.32752739e-09 1.07780646e-13 3.06195899e-16 9.99998689e-01
 5.13075173e-12 3.64593831e-08]
```

Deep network model (with data augmentation)

```
# the deep model also performed better with data augmentation
# this indicates that the augmentation generated more diverse examples
that helped the network to generalize beyond the training data
model_deep = get_deep()
model_deep.fit(augment.flow(X_train, train_y),
validation_data=(X_test, test_y), epochs=epochs
               )

test_loss, test_acc = model_deep.evaluate(X_test, test_y, verbose=2)

print('Accuracy on test set:', test_acc)
```

```
predictions = model_deep.predict(X_test)
print('Predictions for the first image:')
print(predictions[0])
```

```
Epoch 1/10
1875/1875 [==============================] - 48s 25ms/step - loss:
0.5444 - accuracy: 0.8189 - val_loss: 0.0754 - val_accuracy: 0.9789
Epoch 2/10
1875/1875 [==============================] - 46s 25ms/step - loss:
0.1477 - accuracy: 0.9566 - val_loss: 0.0560 - val_accuracy: 0.9837
Epoch 3/10
1875/1875 [==============================] - 47s 25ms/step - loss:
0.1130 - accuracy: 0.9672 - val_loss: 0.0439 - val_accuracy: 0.9881
Epoch 4/10
1875/1875 [==============================] - 47s 25ms/step - loss:
0.0944 - accuracy: 0.9722 - val_loss: 0.0392 - val_accuracy: 0.9893
Epoch 5/10
1875/1875 [==============================] - 47s 25ms/step - loss:
0.0883 - accuracy: 0.9754 - val_loss: 0.0326 - val_accuracy: 0.9903
Epoch 6/10
1875/1875 [==============================] - 47s 25ms/step - loss:
0.0800 - accuracy: 0.9777 - val_loss: 0.0290 - val_accuracy: 0.9919
Epoch 7/10
1875/1875 [==============================] - 48s 25ms/step - loss:
0.0752 - accuracy: 0.9793 - val_loss: 0.0341 - val_accuracy: 0.9903
Epoch 8/10
1875/1875 [==============================] - 48s 26ms/step - loss:
0.0700 - accuracy: 0.9802 - val_loss: 0.0253 - val_accuracy: 0.9927
Epoch 9/10
1875/1875 [==============================] - 47s 25ms/step - loss:
0.0652 - accuracy: 0.9819 - val_loss: 0.0433 - val_accuracy: 0.9874
Epoch 10/10
1875/1875 [==============================] - 47s 25ms/step - loss:
0.0639 - accuracy: 0.9821 - val_loss: 0.0250 - val_accuracy: 0.9924
313/313 - 2s - loss: 0.0250 - accuracy: 0.9924 - 2s/epoch - 5ms/step
Accuracy on test set: 0.9923999905586243
Predictions for the first image:
[1.0166150e-10 1.5316401e-06 2.7028724e-04 5.3955932e-07 6.1020927e-08
 3.0924447e-11 6.3180355e-12 9.9972743e-01 1.3699378e-09 1.2389241e-
07]
```

The best testing accuracy was on the deep netowork model with data augmentation. Accuracy was only slightly better that the other models (approx. 0.001). Since data augmentation tends to prevent overfitting and increase oversampling, it makes sense that the better networks would be one of the networks with data augmentation. It also makes sense that deep networks would perform better, considering that they had way more parameters than the shallow network.