Лабораторная работа №2. Реализация глубокой нейронной сети

1. Реализуйте полносвязную нейронную сеть с помощью библиотеки Tensor Flow. В качестве алгоритма оптимизации можно использовать, например, стохастический градиент (Stochastic Gradient Descent, SGD). Определите количество скрытых слоев от 1 до 5, количество нейронов в каждом из слоев до нескольких сотен, а также их функции активации (кусочно-линейная, сигмоидная, гиперболический тангенс и т.д.).
2. Как улучшилась точность классификатора по сравнению с логистической регрессией?
3. Используйте регуляризацию и метод сброса нейронов (dropout) для борьбы с переобучением. Как улучшилось качество классификации?
4. Воспользуйтесь динамически изменяемой скоростью обучения (learning rate). Наилучшая точность, достигнутая с помощью данной модели составляет 97.1%. Какую точность демонстрирует Ваша реализованная модель?

```
import os
try:
  import wget
except:
  !pip install wget
  import wget
import tarfile


out_dir = 'data/not_mnist'
small_arhive = f'{out_dir}/notMNIST_small.tar.gz'
large_arhive = f'{out_dir}/notMNIST_large.tar.gz'
large_url = 'https://commondatastorage.googleapis.com/books1000/notMNIST_large.tar.gz'
small_url = 'https://commondatastorage.googleapis.com/books1000/notMNIST_small.tar.gz'
```

```
Collecting wget
  Downloading https://files.pythonhosted.org/packages/47/6a/62e288da7bcda82b935ff0c6cfe542970f04e29c756b0e147251b2fb2
Building wheels for collected packages: wget
  Building wheel for wget (setup.py) ... done
  Created wheel for wget: filename=wget-3.2-cp36-none-any.whl size=9682 sha256=50e0f3f3d45a0ce8bfeec8caf1f9529fa273e4
  Stored in directory: /root/.cache/pip/wheels/40/15/30/7d8f7cea2902b4db79e3fea550d7d7b85ecb27ef992b618f3f
Successfully built wget
Installing collected packages: wget
Successfully installed wget-3.2
```

```
if not os.path.exists(out_dir):
  os.makedirs(out_dir)

if not os.path.exists(small_arhive):
  print(f"Downloading {small_arhive}.")
  wget.download(small_url, small_arhive)
  print()
else:
  print(f"Skipping {small_arhive} download (already exists)")

if not os.path.exists(large_arhive):
  print(f"Downloading {large_arhive}.")
  wget.download(large_url, large_arhive)
  print()
else:
  print(f"Skipping {large_arhive} download (already exists)")
```

```
Downloading data/not_mnist/notMNIST_small.tar.gz.

Downloading data/not_mnist/notMNIST_large.tar.gz.
```

```
print(f"Extracting {small_arhive}")
with tarfile.open(small_arhive) as tar:
  tar.extractall(out_dir)

print(f"Extracting {large_arhive}")
```

```
with tarfile.open(large_arhive) as tar:
  tar.extractall(out_dir)
```

```
Extracting data/not_mnist/notMNIST_small.tar.gz
Extracting data/not_mnist/notMNIST_large.tar.gz
```

```
import numpy as np
from pathlib import Path
from PIL import Image

def remove_duplicates(img_train, labels_train, img_test):
    img_new, labels_new = [], []
    test_set = {e.tostring() for e in img_test}
    for i, (x, y) in enumerate(zip(img_train, labels_train)):
        if x.tostring() not in test_set:
            img_new.append(x)
            labels_new.append(y)

    print(f'Removed {img_train.shape[0] - len(img_new)} duplicated images')
    return np.array(img_new), np.array(labels_new)

def load_images(path, n):
    labels = ['I', 'G', 'A', 'F', 'H', 'J', 'C', 'D', 'E', 'B']

    x, y = [], []
    for i, l in enumerate(labels):
        d = Path(path) / l
        print(f'Loading {str(d)} ', end='')
        for j, f in zip(range(n), d.iterdir()):
            try:
                with Image.open(f) as img:
                    x.append(np.array(img))
                    y.append(i)
            except OSError:
                pass
            if j % 1000 == 0:
                print('.', end='', flush=True)
        print(flush=True)
```

```
    return np.array(labels), np.array(x), np.array(y)

def load_not_mnist_data(path='data/not_mnist/', use_cache=True):
    train_folder = Path(path) / 'notMNIST_large'
    test_folder = Path(path) / 'notMNIST_small'

    train_cache_file = Path(path) / 'train.npz'
    test_cache_file = Path(path) / 'test.npz'

    if train_cache_file.exists() and test_cache_file.exists() and use_cache:
        f = np.load(train_cache_file)
        labels, img_train, labels_train = [v for k, v in f.items()]
        f = np.load(test_cache_file)
        labels, img_test, labels_test = [v for k, v in f.items()]
        print('Loaded cached arrays')

    else:
        labels, img_train, labels_train = load_images(train_folder, 10000000)
        labels, img_test, labels_test = load_images(test_folder, 10000000)
        np.savez(train_cache_file, labels, img_train, labels_train)
        np.savez(test_cache_file, labels, img_test, labels_test)

    return labels, img_train, labels_train, img_test, labels_test


labels, img_train, labels_train, img_test, labels_test = load_not_mnist_data()
```

```
Loaded cached arrays
```

```
img_train, labels_train = remove_duplicates(img_train, labels_train, img_test)
```

```
Removed 12213 duplicated images
```

```
from sklearn.linear_model import SGDClassifier, LogisticRegression
from sklearn.model_selection import train_test_split

def flatten(a):
    return a.reshape(a.shape[0], a.shape[1] * a.shape[2])
```

```
    return a.reshape(a.shape[0], a.shape[1] * a.shape[2])

def load_data():
    labels, img_train, labels_train, img_test, labels_test = load_not_mnist_data()
    img_train, labels_train = remove_duplicates(img_train, labels_train, img_test)
    return labels, flatten(img_train), labels_train, flatten(img_test), labels_test



labels, img_train, labels_train, img_test, labels_test = load_not_mnist_data()
img_train, labels_train = remove_duplicates(img_train, labels_train, img_test)
```

```
Loaded cached arrays
Removed 12213 duplicated images
```

Задание 1. Реализуйте полносвязную нейронную сеть с помощью библиотеки Tensor Flow. В качестве алгоритма оптимизации можно использовать, например, стохастический градиент (Stochastic Gradient Descent, SGD). Определите количество скрытых слоев от 1 до 5, количество нейронов в каждом из слоев до нескольких сотен, а также их функции активации (кусочно-линейная, сигмоидная, гиперболический тангенс и т.д.).

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
from keras.callbacks import EarlyStopping

results = {}
results.setdefault('val_acc', {})

model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(200, activation='relu'),
    tf.keras.layers.Dense(200, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
```

```
                metrics=['accuracy'])
callback = tf.keras.callbacks.EarlyStopping(monitor='accuracy', patience=3)
model.fit(x=img_train, y=labels_train, epochs=100,
          callbacks=[callback], validation_split=0.1)
print('\n# Evaluate')
result = model.evaluate(img_test, labels_test)
print(result)
```

```
Epoch 1/100
16154/16154 [==============================] - 44s 3ms/step - loss: 0.9557 - accuracy: 0.7816
Epoch 2/100
16154/16154 [==============================] - 43s 3ms/step - loss: 0.5695 - accuracy: 0.8335
Epoch 3/100
16154/16154 [==============================] - 43s 3ms/step - loss: 0.5467 - accuracy: 0.8385
Epoch 4/100
16154/16154 [==============================] - 42s 3ms/step - loss: 0.5205 - accuracy: 0.8465
Epoch 5/100
16154/16154 [==============================] - 44s 3ms/step - loss: 0.5058 - accuracy: 0.8506
Epoch 6/100
16154/16154 [==============================] - 43s 3ms/step - loss: 0.4956 - accuracy: 0.8537
Epoch 7/100
16154/16154 [==============================] - 43s 3ms/step - loss: 0.4864 - accuracy: 0.8555
Epoch 8/100
16154/16154 [==============================] - 43s 3ms/step - loss: 0.4766 - accuracy: 0.8586
Epoch 9/100
16154/16154 [==============================] - 43s 3ms/step - loss: 0.4686 - accuracy: 0.8604
Epoch 10/100
16154/16154 [==============================] - 42s 3ms/step - loss: 0.4629 - accuracy: 0.8621
Epoch 11/100
16154/16154 [==============================] - 42s 3ms/step - loss: 0.4588 - accuracy: 0.8636
Epoch 12/100
16154/16154 [==============================] - 43s 3ms/step - loss: 0.4567 - accuracy: 0.8637
Epoch 13/100
16154/16154 [==============================] - 42s 3ms/step - loss: 0.4522 - accuracy: 0.8645
Epoch 14/100
16154/16154 [==============================] - 44s 3ms/step - loss: 0.4501 - accuracy: 0.8649
Epoch 15/100
16154/16154 [==============================] - 42s 3ms/step - loss: 0.4463 - accuracy: 0.8659
Epoch 16/100
16154/16154 [==============================] - 43s 3ms/step - loss: 0.4474 - accuracy: 0.8664
Epoch 17/100
16154/16154 [==============================] - 43s 3ms/step - loss: 0.4408 - accuracy: 0.8674
Epoch 18/100
16154/16154 [==============================] - 42s 3ms/step - loss: 0.4404 - accuracy: 0.8681
Epoch 19/100
16154/16154 [==============================] - 43s 3ms/step - loss: 0.4389 - accuracy: 0.8681
Epoch 20/100
16154/16154 [==============================] - 44s 3ms/step - loss: 0.4352 - accuracy: 0.8690
Epoch 21/100
16154/16154 [==============================] - 43s 3ms/step - loss: 0.4349 - accuracy: 0.8693
```

```
Epoch 22/100
16154/16154 [==============================] - 43s 3ms/step - loss: 0.4317 - accuracy: 0.8693
Epoch 23/100
16154/16154 [==============================] - 42s 3ms/step - loss: 0.4313 - accuracy: 0.8698
Epoch 24/100
16154/16154 [==============================] - 42s 3ms/step - loss: 0.4301 - accuracy: 0.8702
Epoch 25/100
16154/16154 [==============================] - 42s 3ms/step - loss: 0.4297 - accuracy: 0.8704
Epoch 26/100
16154/16154 [==============================] - 42s 3ms/step - loss: 0.4299 - accuracy: 0.8707
Epoch 27/100
16154/16154 [==============================] - 42s 3ms/step - loss: 0.4275 - accuracy: 0.8713
Epoch 28/100
16154/16154 [==============================] - 42s 3ms/step - loss: 0.4276 - accuracy: 0.8708
Epoch 29/100
16154/16154 [==============================] - 43s 3ms/step - loss: 0.4268 - accuracy: 0.8720
Epoch 30/100
16154/16154 [==============================] - 42s 3ms/step - loss: 0.4240 - accuracy: 0.8722
Epoch 31/100
16154/16154 [==============================] - 42s 3ms/step - loss: 0.4239 - accuracy: 0.8726
Epoch 32/100
16154/16154 [==============================] - 43s 3ms/step - loss: 0.4218 - accuracy: 0.8725
Epoch 33/100
16154/16154 [==============================] - 42s 3ms/step - loss: 0.4207 - accuracy: 0.8729
Epoch 34/100
16154/16154 [==============================] - 42s 3ms/step - loss: 0.4209 - accuracy: 0.8732
Epoch 35/100
16154/16154 [==============================] - 42s 3ms/step - loss: 0.4172 - accuracy: 0.8735
Epoch 36/100
16154/16154 [==============================] - 43s 3ms/step - loss: 0.4179 - accuracy: 0.8739
Epoch 37/100
16154/16154 [==============================] - 42s 3ms/step - loss: 0.4176 - accuracy: 0.8736
Epoch 38/100
16154/16154 [==============================] - 42s 3ms/step - loss: 0.4184 - accuracy: 0.8736
Epoch 39/100
16154/16154 [==============================] - 43s 3ms/step - loss: 0.4157 - accuracy: 0.8738

# Evaluate
586/586 [==============================] - 1s 1ms/step - loss: 0.2734 - accuracy: 0.9323
[0.27337297797203064, 0.9323328137397766]
```

Задание 2. Как улучшилась точность классификатора по сравнению с логистической регрессией?

Точность классификатора повысилась на 11 процентов

Задание 3. Используйте регуляризацию и метод сброса нейронов (dropout) для борьбы с переобучением. Как улучшилось качество классификации?

В моей модели качество классификации не улучшилось при добавлении, скорее всего штраф был большим, и отбрасывались веса, которые могли бы описать удачнее модель, пытался уменьшать штраф, так как результат был еще меньше чем без регуляризации

```
initializer = tf.keras.initializers.TruncatedNormal(mean=0.0, stddev=0.05, seed=None)
regL1 = tf.keras.regularizers.l1(0.0001)
regL2 = tf.keras.regularizers.l2(0.0001)
model2 = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(200,
                          activation='relu',
                          kernel_regularizer=regL2,
                          bias_regularizer=regL2,
                          bias_initializer=initializer,
                          kernel_initializer=initializer),
    tf.keras.layers.Dense(200,
                          activation='relu',
                          kernel_regularizer=regL2,
                          bias_regularizer=regL2,
                          bias_initializer=initializer,
                          kernel_initializer=initializer),
    tf.keras.layers.Dense(10, activation='softmax')
])

model2.compile(optimizer='adam',
               loss='sparse_categorical_crossentropy',
               metrics=['accuracy'])

#steps_per_epoch=tf.math.ceil(train_size/BATCH_SIZE).numpy()
#model.fit(train_dataset, epochs=10, steps_per_epoch=steps_per_epoch, validation_data=(test_dataset))
```

```
model2.fit(x=img_train, y=labels_train, epochs=100,
          callbacks=[callback], validation_split=0.1)
#model.fit(train_dataset, epochs=2, steps_per_epoch=steps_per_epoch)
print('\n# Evaluate')
result2 = model2.evaluate(img_test, labels_test)
print (result2)
```

```
Epoch 1/100
16154/16154 [==============================] - 50s 3ms/step - loss: 0.7922 - accuracy: 0.8100
Epoch 2/100
16154/16154 [==============================] - 49s 3ms/step - loss: 0.5592 - accuracy: 0.8408
Epoch 3/100
16154/16154 [==============================] - 50s 3ms/step - loss: 0.5473 - accuracy: 0.8456
Epoch 4/100
16154/16154 [==============================] - 49s 3ms/step - loss: 0.5333 - accuracy: 0.8509
Epoch 5/100
16154/16154 [==============================] - 50s 3ms/step - loss: 0.5266 - accuracy: 0.8529
Epoch 6/100
16154/16154 [==============================] - 48s 3ms/step - loss: 0.5244 - accuracy: 0.8544
Epoch 7/100
16154/16154 [==============================] - 48s 3ms/step - loss: 0.5219 - accuracy: 0.8550
Epoch 8/100
16154/16154 [==============================] - 49s 3ms/step - loss: 0.5200 - accuracy: 0.8562
Epoch 9/100
16154/16154 [==============================] - 49s 3ms/step - loss: 0.5197 - accuracy: 0.8566
Epoch 10/100
16154/16154 [==============================] - 50s 3ms/step - loss: 0.5173 - accuracy: 0.8576
Epoch 11/100
16154/16154 [==============================] - 49s 3ms/step - loss: 0.5178 - accuracy: 0.8568
Epoch 12/100
16154/16154 [==============================] - 49s 3ms/step - loss: 0.5184 - accuracy: 0.8573
Epoch 13/100
16154/16154 [==============================] - 53s 3ms/step - loss: 0.5169 - accuracy: 0.8576
Epoch 14/100
16154/16154 [==============================] - 55s 3ms/step - loss: 0.5170 - accuracy: 0.8572
Epoch 15/100
16154/16154 [==============================] - 53s 3ms/step - loss: 0.5171 - accuracy: 0.8572
Epoch 16/100
16154/16154 [==============================] - 53s 3ms/step - loss: 0.5175 - accuracy: 0.8572

# Evaluate
586/586 [==============================] - 1s 2ms/step - loss: 0.3239 - accuracy: 0.9172
[0.3239336907863617, 0.9172185659408569]
```

```
model3 = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
```

```
    tf.keras.layers.Dense(200,
                          activation='relu',
                          kernel_regularizer=regL2,
                          bias_regularizer=regL2,
                          bias_initializer=initializer,
                          kernel_initializer=initializer),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(200,
                          activation='relu',
                          kernel_regularizer=regL2,
                          bias_regularizer=regL2,
                          bias_initializer=initializer,
                          kernel_initializer=initializer),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(10, activation='softmax')
])

model3.compile(optimizer='adam',
               loss='sparse_categorical_crossentropy',
               metrics=['accuracy'])

#steps_per_epoch=tf.math.ceil(train_size/BATCH_SIZE).numpy()
#model.fit(train_dataset, epochs=10, steps_per_epoch=steps_per_epoch, validation_data=(test_dataset))
model3.fit(x=img_train, y=labels_train, epochs=100,
           callbacks=[callback], validation_split=0.1)
#model.fit(train_dataset, epochs=2, steps_per_epoch=steps_per_epoch)
print('\n# Evaluate')
result = model3.evaluate(img_test, labels_test)
print (result)
```

```
Epoch 1/100
16154/16154 [==============================] - 50s 3ms/step - loss: 1.0940 - accuracy: 0.7254
Epoch 2/100
16154/16154 [==============================] - 50s 3ms/step - loss: 0.6626 - accuracy: 0.8030
Epoch 3/100
16154/16154 [==============================] - 49s 3ms/step - loss: 0.6378 - accuracy: 0.8119
Epoch 4/100
16154/16154 [==============================] - 49s 3ms/step - loss: 0.6360 - accuracy: 0.8107
Epoch 5/100
16154/16154 [==============================] - 49s 3ms/step - loss: 0.6263 - accuracy: 0.8154
Epoch 6/100
16154/16154 [==============================] - 50s 3ms/step - loss: 0.6150 - accuracy: 0.8178
Epoch 7/100
16154/16154 [==============================] - 49s 3ms/step - loss: 0.6083 - accuracy: 0.8191
Epoch 8/100
16154/16154 [==============================] - 48s 3ms/step - loss: 0.6029 - accuracy: 0.8211
Epoch 9/100
16154/16154 [==============================] - 49s 3ms/step - loss: 0.5996 - accuracy: 0.8218
Epoch 10/100
16154/16154 [==============================] - 49s 3ms/step - loss: 0.5962 - accuracy: 0.8232
Epoch 11/100
16154/16154 [==============================] - 49s 3ms/step - loss: 0.5903 - accuracy: 0.8244
Epoch 12/100
16154/16154 [==============================] - 49s 3ms/step - loss: 0.5894 - accuracy: 0.8248
Epoch 13/100
16154/16154 [==============================] - 50s 3ms/step - loss: 0.5824 - accuracy: 0.8263
Epoch 14/100
16154/16154 [==============================] - 51s 3ms/step - loss: 0.5805 - accuracy: 0.8262
Epoch 15/100
16154/16154 [==============================] - 51s 3ms/step - loss: 0.5777 - accuracy: 0.8281
Epoch 16/100
16154/16154 [==============================] - 51s 3ms/step - loss: 0.5761 - accuracy: 0.8284
Epoch 17/100
16154/16154 [==============================] - 51s 3ms/step - loss: 0.5718 - accuracy: 0.8296
Epoch 18/100
16154/16154 [==============================] - 51s 3ms/step - loss: 0.5705 - accuracy: 0.8299
Epoch 19/100
16154/16154 [==============================] - 51s 3ms/step - loss: 0.5681 - accuracy: 0.8308
Epoch 20/100
16154/16154 [==============================] - 50s 3ms/step - loss: 0.5646 - accuracy: 0.8322
Epoch 21/100
16154/16154 [==============================] - 49s 3ms/step - loss: 0.5633 - accuracy: 0.8324
```

```
Epoch 22/100
16154/16154 [==============================] - 50s 3ms/step - loss: 0.5681 - accuracy: 0.8318
Epoch 23/100
16154/16154 [==============================] - 50s 3ms/step - loss: 0.5593 - accuracy: 0.8339
Epoch 24/100
16154/16154 [==============================] - 50s 3ms/step - loss: 0.5604 - accuracy: 0.8337
Epoch 25/100
16154/16154 [==============================] - 52s 3ms/step - loss: 0.5576 - accuracy: 0.8358
Epoch 26/100
16154/16154 [==============================] - 52s 3ms/step - loss: 0.5569 - accuracy: 0.8361
Epoch 27/100
16154/16154 [==============================] - 50s 3ms/step - loss: 0.5556 - accuracy: 0.8364
Epoch 28/100
16154/16154 [==============================] - 54s 3ms/step - loss: 0.5524 - accuracy: 0.8375
Epoch 29/100
16154/16154 [==============================] - 59s 4ms/step - loss: 0.5566 - accuracy: 0.8376
Epoch 30/100
16154/16154 [==============================] - 55s 3ms/step - loss: 0.5534 - accuracy: 0.8374
Epoch 31/100
16154/16154 [==============================] - 59s 4ms/step - loss: 0.5506 - accuracy: 0.8376
Epoch 32/100
16154/16154 [==============================] - 54s 3ms/step - loss: 0.5474 - accuracy: 0.8389
Epoch 33/100
16154/16154 [==============================] - 55s 3ms/step - loss: 0.5489 - accuracy: 0.8385
Epoch 34/100
16154/16154 [==============================] - 55s 3ms/step - loss: 0.5483 - accuracy: 0.8380
Epoch 35/100
16154/16154 [==============================] - 55s 3ms/step - loss: 0.5479 - accuracy: 0.8378

# Evaluate
586/586 [==============================] - 1s 1ms/step - loss: 0.2875 - accuracy: 0.9201
[0.28747716546058655, 0.9201025366783142]
```

Задание 4. Воспользуйтесь динамически изменяемой скоростью обучения (learning rate). Наилучшая точность, достигнутая с помощью данной модели составляет 97.1%. Какую точность демонстрирует Ваша реализованная модель?

Сравнение алгоритма без динамического шага, все предыдущие были с оптимизатором adam

```
model4 = tf.keras.Sequential([
```

```
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(300, activation='relu'),
    tf.keras.layers.Dense(300, activation='relu'),
    tf.keras.layers.Dense(300, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])
learning_rate = 0.001
optimizer = tf.compat.v1.train.GradientDescentOptimizer(learning_rate)
model4.compile(optimizer=optimizer,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

#steps_per_epoch=tf.math.ceil(train_size/BATCH_SIZE).numpy()
#model.fit(train_dataset, epochs=10, steps_per_epoch=steps_per_epoch, validation_data=(test_dataset))
model4.fit(x=img_train, y=labels_train, epochs=100,
          callbacks=[callback], validation_split=0.1)
#model.fit(train_dataset, epochs=2, steps_per_epoch=steps_per_epoch)
print('\n# Evaluate')
result = model4.evaluate(img_test, labels_test)
print (result)
```

```
Epoch 1/100
14538/14538 [==============================] - 76s 5ms/step - loss: 0.7099 - accuracy: 0.8281 - val_loss: 3.9638 - va
Epoch 2/100
14538/14538 [==============================] - 76s 5ms/step - loss: 0.4403 - accuracy: 0.8622 - val_loss: 3.0196 - va
Epoch 3/100
14538/14538 [==============================] - 74s 5ms/step - loss: 0.4049 - accuracy: 0.8737 - val_loss: 3.0639 - va
Epoch 4/100
14538/14538 [==============================] - 74s 5ms/step - loss: 0.3809 - accuracy: 0.8807 - val_loss: 2.5569 - va
Epoch 5/100
14538/14538 [==============================] - 78s 5ms/step - loss: 0.3631 - accuracy: 0.8860 - val_loss: 2.8790 - va
Epoch 6/100
14538/14538 [==============================] - 76s 5ms/step - loss: 0.3487 - accuracy: 0.8904 - val_loss: 3.0503 - va
Epoch 7/100
14538/14538 [==============================] - 73s 5ms/step - loss: 0.3365 - accuracy: 0.8938 - val_loss: 2.8967 - va
Epoch 8/100
14538/14538 [==============================] - 73s 5ms/step - loss: 0.3255 - accuracy: 0.8972 - val_loss: 2.2211 - va
Epoch 9/100
14538/14538 [==============================] - 74s 5ms/step - loss: 0.3164 - accuracy: 0.8998 - val_loss: 2.7794 - va
Epoch 10/100
14538/14538 [==============================] - 74s 5ms/step - loss: 0.3079 - accuracy: 0.9027 - val_loss: 2.3318 - va
Epoch 11/100
14538/14538 [==============================] - 72s 5ms/step - loss: 0.2996 - accuracy: 0.9049 - val_loss: 2.6329 - va
Epoch 12/100
14538/14538 [==============================] - 72s 5ms/step - loss: 0.2924 - accuracy: 0.9070 - val_loss: 2.4085 - va
Epoch 13/100
14538/14538 [==============================] - 73s 5ms/step - loss: 0.2856 - accuracy: 0.9089 - val_loss: 2.5572 - va
Epoch 14/100
14538/14538 [==============================] - 73s 5ms/step - loss: 0.2795 - accuracy: 0.9109 - val_loss: 2.6375 - va
Epoch 15/100
14538/14538 [==============================] - 75s 5ms/step - loss: 0.2734 - accuracy: 0.9124 - val_loss: 2.2545 - va
Epoch 16/100
14538/14538 [==============================] - 72s 5ms/step - loss: 0.2678 - accuracy: 0.9142 - val_loss: 2.4537 - va
Epoch 17/100
14538/14538 [==============================] - 73s 5ms/step - loss: 0.2624 - accuracy: 0.9160 - val_loss: 2.5731 - va
Epoch 18/100
14538/14538 [==============================] - 73s 5ms/step - loss: 0.2570 - accuracy: 0.9176 - val_loss: 2.4875 - va
Epoch 19/100
14538/14538 [==============================] - 73s 5ms/step - loss: 0.2524 - accuracy: 0.9190 - val_loss: 2.6938 - va
Epoch 20/100
14538/14538 [==============================] - 74s 5ms/step - loss: 0.2476 - accuracy: 0.9205 - val_loss: 2.6638 - va
Epoch 21/100
14538/14538 [==============================] - 74s 5ms/step - loss: 0.2433 - accuracy: 0.9219 - val_loss: 2.7956 - va
```

```
Epoch 22/100
14538/14538 [==============================] - 73s 5ms/step - loss: 0.2388 - accuracy: 0.9231 - val_loss: 2.6273 - va
Epoch 23/100
14538/14538 [==============================] - 74s 5ms/step - loss: 0.2346 - accuracy: 0.9247 - val_loss: 2.5814 - va
Epoch 24/100
14538/14538 [==============================] - 73s 5ms/step - loss: 0.2308 - accuracy: 0.9259 - val_loss: 2.5701 - va
Epoch 25/100
14538/14538 [==============================] - 72s 5ms/step - loss: 0.2266 - accuracy: 0.9272 - val_loss: 2.6839 - va
Epoch 26/100
14538/14538 [==============================] - 73s 5ms/step - loss: 0.2226 - accuracy: 0.9284 - val_loss: 2.9174 - va
Epoch 27/100
14538/14538 [==============================] - 71s 5ms/step - loss: 0.2192 - accuracy: 0.9292 - val_loss: 2.5698 - va
Epoch 28/100
14538/14538 [==============================] - 73s 5ms/step - loss: 0.2159 - accuracy: 0.9301 - val_loss: 2.5828 - va
Epoch 29/100
14538/14538 [==============================] - 74s 5ms/step - loss: 0.2122 - accuracy: 0.9314 - val_loss: 2.6292 - va
Epoch 30/100
14538/14538 [==============================] - 75s 5ms/step - loss: 0.2091 - accuracy: 0.9324 - val_loss: 2.6492 - va
Epoch 31/100
14538/14538 [==============================] - 75s 5ms/step - loss: 0.2054 - accuracy: 0.9336 - val_loss: 3.1578 - va
Epoch 32/100
14538/14538 [==============================] - 74s 5ms/step - loss: 0.2024 - accuracy: 0.9346 - val_loss: 2.7782 - va
Epoch 33/100
14538/14538 [==============================] - 72s 5ms/step - loss: 0.1992 - accuracy: 0.9355 - val_loss: 2.8469 - va
Epoch 34/100
14538/14538 [==============================] - 73s 5ms/step - loss: 0.1960 - accuracy: 0.9364 - val_loss: 2.7799 - va
Epoch 35/100
14538/14538 [==============================] - 73s 5ms/step - loss: 0.1929 - accuracy: 0.9376 - val_loss: 3.1547 - va
Epoch 36/100
14538/14538 [==============================] - 72s 5ms/step - loss: 0.1904 - accuracy: 0.9381 - val_loss: 2.9929 - va
Epoch 37/100
14538/14538 [==============================] - 73s 5ms/step - loss: 0.1872 - accuracy: 0.9392 - val_loss: 2.8968 - va
Epoch 38/100
14538/14538 [==============================] - 72s 5ms/step - loss: 0.1848 - accuracy: 0.9400 - val_loss: 3.2073 - va
Epoch 39/100
14538/14538 [==============================] - 72s 5ms/step - loss: 0.1824 - accuracy: 0.9407 - val_loss: 3.2207 - va
Epoch 40/100
14538/14538 [==============================] - 73s 5ms/step - loss: 0.1794 - accuracy: 0.9417 - val_loss: 3.0908 - va
Epoch 41/100
14538/14538 [==============================] - 73s 5ms/step - loss: 0.1769 - accuracy: 0.9424 - val_loss: 2.9356 - va
Epoch 42/100
14538/14538 [==============================] - 71s 5ms/step - loss: 0.1744 - accuracy: 0.9434 - val_loss: 3.0271 - va
```

```
Epoch 43/100
14538/14538 [==============================] - 71s 5ms/step - loss: 0.1720 - accuracy: 0.9440 - val_loss: 3.0928 - va
Epoch 44/100
14538/14538 [==============================] - 72s 5ms/step - loss: 0.1696 - accuracy: 0.9448 - val_loss: 3.0603 - va
Epoch 45/100
14538/14538 [==============================] - 73s 5ms/step - loss: 0.1675 - accuracy: 0.9456 - val_loss: 3.7177 - va
Epoch 46/100
14538/14538 [==============================] - 72s 5ms/step - loss: 0.1652 - accuracy: 0.9461 - val_loss: 3.0598 - va
Epoch 47/100
14538/14538 [==============================] - 73s 5ms/step - loss: 0.1629 - accuracy: 0.9470 - val_loss: 3.6798 - va
Epoch 48/100
14538/14538 [==============================] - 73s 5ms/step - loss: 0.1608 - accuracy: 0.9478 - val_loss: 3.4252 - va
Epoch 49/100
14538/14538 [==============================] - 75s 5ms/step - loss: 0.1584 - accuracy: 0.9483 - val_loss: 3.4208 - va
Epoch 50/100
14538/14538 [==============================] - 73s 5ms/step - loss: 0.1566 - accuracy: 0.9492 - val_loss: 3.5609 - va
Epoch 51/100
14538/14538 [==============================] - 73s 5ms/step - loss: 0.1547 - accuracy: 0.9494 - val_loss: 3.2792 - va
Epoch 52/100
14538/14538 [==============================] - 73s 5ms/step - loss: 0.1528 - accuracy: 0.9501 - val_loss: 3.7867 - va
Epoch 53/100
14538/14538 [==============================] - 72s 5ms/step - loss: 0.1505 - accuracy: 0.9508 - val_loss: 3.6727 - va
Epoch 54/100
14538/14538 [==============================] - 75s 5ms/step - loss: 0.1486 - accuracy: 0.9516 - val_loss: 3.3847 - va
Epoch 55/100
14538/14538 [==============================] - 74s 5ms/step - loss: 0.1472 - accuracy: 0.9517 - val_loss: 3.3044 - va
Epoch 56/100
14538/14538 [==============================] - 73s 5ms/step - loss: 0.1454 - accuracy: 0.9525 - val_loss: 3.6836 - va
Epoch 57/100
14538/14538 [==============================] - 72s 5ms/step - loss: 0.1439 - accuracy: 0.9529 - val_loss: 3.4021 - va
Epoch 58/100
14538/14538 [==============================] - 73s 5ms/step - loss: 0.1414 - accuracy: 0.9537 - val_loss: 3.4399 - va
Epoch 59/100
14538/14538 [==============================] - 71s 5ms/step - loss: 0.1401 - accuracy: 0.9542 - val_loss: 3.4956 - va
Epoch 60/100
14538/14538 [==============================] - 72s 5ms/step - loss: 0.1382 - accuracy: 0.9546 - val_loss: 3.7009 - va
Epoch 61/100
14538/14538 [==============================] - 72s 5ms/step - loss: 0.1372 - accuracy: 0.9549 - val_loss: 4.0359 - va
Epoch 62/100
14538/14538 [==============================] - 76s 5ms/step - loss: 0.1353 - accuracy: 0.9557 - val_loss: 4.0258 - va
Epoch 63/100
14538/14538 [==============================] - 71s 5ms/step - loss: 0.1336 - accuracy: 0.9562 - val_loss: 3.5179 - va
Epoch 64/100
```

```
Epoch 64/100
14538/14538 [==============================] - 66s 5ms/step - loss: 0.1324 - accuracy: 0.9567 - val_loss: 3.8658 - va
Epoch 65/100
14538/14538 [==============================] - 66s 5ms/step - loss: 0.1306 - accuracy: 0.9570 - val_loss: 4.0361 - va
Epoch 66/100
14538/14538 [==============================] - 67s 5ms/step - loss: 0.1296 - accuracy: 0.9574 - val_loss: 3.9768 - va
Epoch 67/100
14538/14538 [==============================] - 68s 5ms/step - loss: 0.1278 - accuracy: 0.9581 - val_loss: 3.8589 - va
Epoch 68/100
14538/14538 [==============================] - 67s 5ms/step - loss: 0.1262 - accuracy: 0.9589 - val_loss: 4.1609 - va
Epoch 69/100
14538/14538 [==============================] - 66s 5ms/step - loss: 0.1249 - accuracy: 0.9589 - val_loss: 4.4513 - va
Epoch 70/100
14538/14538 [==============================] - 67s 5ms/step - loss: 0.1235 - accuracy: 0.9595 - val_loss: 3.7937 - va
Epoch 71/100
14538/14538 [==============================] - 68s 5ms/step - loss: 0.1223 - accuracy: 0.9598 - val_loss: 3.8595 - va
Epoch 72/100
14538/14538 [==============================] - 72s 5ms/step - loss: 0.1212 - accuracy: 0.9602 - val_loss: 3.9233 - va
Epoch 73/100
14538/14538 [==============================] - 69s 5ms/step - loss: 0.1201 - accuracy: 0.9605 - val_loss: 3.8487 - va
Epoch 74/100
14538/14538 [==============================] - 68s 5ms/step - loss: 0.1186 - accuracy: 0.9610 - val_loss: 4.6559 - va
Epoch 75/100
14538/14538 [==============================] - 69s 5ms/step - loss: 0.1180 - accuracy: 0.9613 - val_loss: 4.2207 - va
Epoch 76/100
14538/14538 [==============================] - 70s 5ms/step - loss: 0.1170 - accuracy: 0.9615 - val_loss: 3.9807 - va
Epoch 77/100
14538/14538 [==============================] - 69s 5ms/step - loss: 0.1152 - accuracy: 0.9620 - val_loss: 4.1908 - va
Epoch 78/100
14538/14538 [==============================] - 70s 5ms/step - loss: 0.1139 - accuracy: 0.9625 - val_loss: 4.3981 - va
Epoch 79/100
14538/14538 [==============================] - 70s 5ms/step - loss: 0.1135 - accuracy: 0.9625 - val_loss: 3.9252 - va
Epoch 80/100
14538/14538 [==============================] - 69s 5ms/step - loss: 0.1125 - accuracy: 0.9629 - val_loss: 4.5508 - va
Epoch 81/100
14538/14538 [==============================] - 69s 5ms/step - loss: 0.1112 - accuracy: 0.9634 - val_loss: 4.4530 - va
Epoch 82/100
14538/14538 [==============================] - 69s 5ms/step - loss: 0.1096 - accuracy: 0.9640 - val_loss: 4.1170 - va
Epoch 83/100
14538/14538 [==============================] - 68s 5ms/step - loss: 0.1083 - accuracy: 0.9644 - val_loss: 4.9726 - va
Epoch 84/100
14538/14538 [==============================] - 69s 5ms/step - loss: 0.1084 - accuracy: 0.9644 - val_loss: 4.3088 - va
Epoch 85/100
```

```
14538/14538 [==============================] - 71s 5ms/step - loss: 0.1074 - accuracy: 0.9647 - val_loss: 4.2204 - va
Epoch 86/100
14538/14538 [==============================] - 71s 5ms/step - loss: 0.1064 - accuracy: 0.9649 - val_loss: 4.5704 - va
Epoch 87/100
14538/14538 [==============================] - 72s 5ms/step - loss: 0.1053 - accuracy: 0.9655 - val_loss: 4.2248 - va
Epoch 88/100
14538/14538 [==============================] - 74s 5ms/step - loss: 0.1040 - accuracy: 0.9656 - val_loss: 4.3814 - va
Epoch 89/100
14538/14538 [==============================] - 72s 5ms/step - loss: 0.1031 - accuracy: 0.9661 - val_loss: 4.7207 - va
Epoch 90/100
14538/14538 [==============================] - 70s 5ms/step - loss: 0.1021 - accuracy: 0.9666 - val_loss: 4.8430 - va
Epoch 91/100
14538/14538 [==============================] - 70s 5ms/step - loss: 0.1018 - accuracy: 0.9664 - val_loss: 4.7164 - va
Epoch 92/100
14538/14538 [==============================] - 70s 5ms/step - loss: 0.1005 - accuracy: 0.9670 - val_loss: 4.6680 - va
Epoch 93/100
14538/14538 [==============================] - 70s 5ms/step - loss: 0.1000 - accuracy: 0.9670 - val_loss: 5.0124 - va
Epoch 94/100
14538/14538 [==============================] - 72s 5ms/step - loss: 0.0990 - accuracy: 0.9673 - val_loss: 4.7769 - va
Epoch 95/100
14538/14538 [==============================] - 70s 5ms/step - loss: 0.0985 - accuracy: 0.9676 - val_loss: 5.0307 - va
Epoch 96/100
14538/14538 [==============================] - 70s 5ms/step - loss: 0.0981 - accuracy: 0.9678 - val_loss: 4.9359 - va
Epoch 97/100
14538/14538 [==============================] - 71s 5ms/step - loss: 0.0964 - accuracy: 0.9683 - val_loss: 4.6103 - va
Epoch 98/100
14538/14538 [==============================] - 71s 5ms/step - loss: 0.0946 - accuracy: 0.9689 - val_loss: 5.0675 - va
Epoch 99/100
14538/14538 [==============================] - 65s 4ms/step - loss: 0.0952 - accuracy: 0.9687 - val_loss: 4.9389 - va
Epoch 100/100
14538/14538 [==============================] - 64s 4ms/step - loss: 0.0953 - accuracy: 0.9684 - val_loss: 5.1727 - va

# Evaluate
586/586 [==============================] - 1s 2ms/step - loss: 0.6750 - accuracy: 0.9272
[0.6749985814094543, 0.9272057414054871]
```