Лабораторная работа №6. Применение сверточных нейронных сетей (многоклассовая классификация)

In [1]:

```
# TensorFlow и tf.keras
import tensorflow as tf
from tensorflow import keras
from keras import regularizers

import numpy as np
import matplotlib.pyplot as plt
import pdb
import os
import scipy.io
from sklearn.model_selection import train_test_split
import tarfile
from six.moves import cPickle as pickle
import zipfile
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
Using TensorFlow backend.
```

Задание 1. Загрузите данные. Разделите исходный набор данных на обучающую и валидационную выборки.

In [2]:

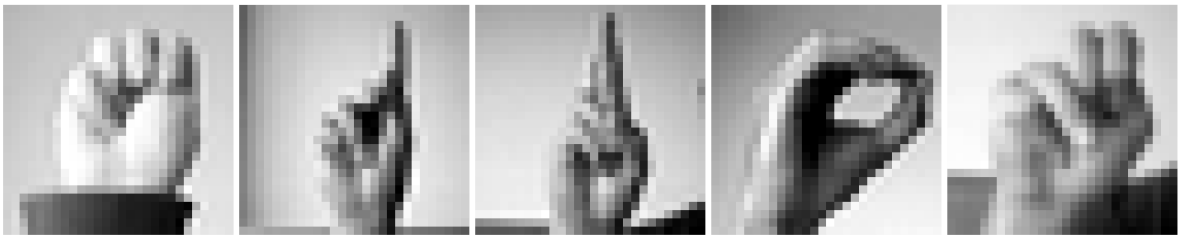
```
def read_images(filename):
    images = []
    with open(filename) as f:
        f.readline()
        for line in f:
            label, *values = line.strip().split(',')
            image = np.array([float(v) for v in values]) / 255
            image.resize((28, 28, 1))
            images.append((int(label), image))

    labels = [p[0] for p in images]
    images = [p[1] for p in images]
    return np.array(labels), np.array(images)

test_labels, test_images = read_images("../data/sign_mnist/sign_mnist_test.csv")
train_labels, train_images = read_images("../data/sign_mnist/sign_mnist_train.csv")
train_images, valid_images, train_labels, valid_labels = train_test_split(train_ima

classes = sorted(list(set(list(train_labels) + list(valid_labels) + list(test_label

print(len(train_images), len(train_labels), len(test_images), len(test_labels), len
```

```
20591 20591 7172 7172 6864 6864
```

In [3]:

```
def plot_samples(images, sample_size, name):
    figure, axes = plt.subplots(1, sample_size, figsize=(28, 28))
    figure.suptitle(name)
    axes = axes.flatten()
    imgs = images[:sample_size]
    for img, ax in zip(imgs, axes):
        img = np.array(img)
        img.resize((28, 28))
        ax.imshow(img, cmap='gray')
        ax.axis('off')
    plt.tight_layout()
    plt.show()

plot_samples(train_images, 5, 'train')
plot_samples(valid_images, 5, 'valid')
plot_samples(test_images, 5, 'test')
```

train



valid

test

Задание 2. Реализуйте глубокую нейронную сеть со сверточными слоями. Какое качество классификации получено? Какая архитектура сети была использована?

In [4]:

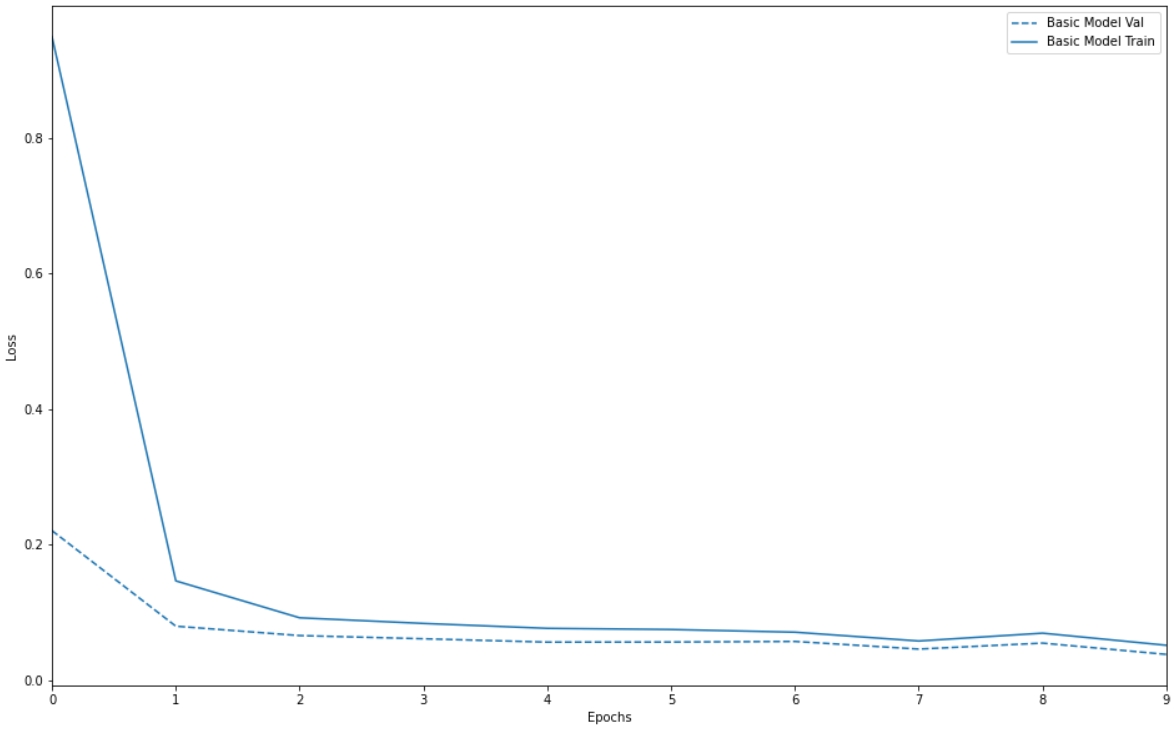
```
l2_regularization = 1e-4
num_classes = max(classes) + 1

basic_model = keras.Sequential([
    keras.layers.Conv2D(32, 3, activation='relu', padding='same', input_shape=(28,
    keras.layers.MaxPooling2D(pool_size=(2, 2)),

    keras.layers.Conv2D(64, 3, activation='relu', padding='same'),
    keras.layers.MaxPooling2D(pool_size=(2, 2)),

    keras.layers.Flatten(),
    keras.layers.Dense(units=512, activation='relu', kernel_regularizer=regularizer
    keras.layers.Dropout(0.25),

    keras.layers.Dense(units=num_classes, activation = 'softmax')
])

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

basic_model.summary()
basic_model_history = basic_model.fit(train_images, train_labels, epochs=10, valida
test_loss, test_acc = basic_model.evaluate(test_images,  test_labels, verbose=2)

print('\nТочность на проверочных данных:', test_acc)
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 3136)              0
_________________________________________________________________
dense (Dense)                (None, 512)               1606144
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 25)                12825
=================================================================
Total params: 1,637,785
Trainable params: 1,637,785
Non-trainable params: 0
_________________________________________________________________
Train on 20591 samples, validate on 6864 samples
Epoch 1/10
20591/20591 [==============================] - 23s 1ms/sample - loss:
0.9484 - accuracy: 0.7231 - val_loss: 0.2210 - val_accuracy: 0.9470
Epoch 2/10
20591/20591 [==============================] - 25s 1ms/sample - loss:
0.1470 - accuracy: 0.9763 - val_loss: 0.0802 - val_accuracy: 0.9983
```

```
Epoch 3/10
20591/20591 [==============================] - 23s 1ms/sample - loss:
0.0926 - accuracy: 0.9924 - val_loss: 0.0664 - val_accuracy: 0.9997
Epoch 4/10
20591/20591 [==============================] - 22s 1ms/sample - loss:
0.0843 - accuracy: 0.9933 - val_loss: 0.0617 - val_accuracy: 0.9997
Epoch 5/10
20591/20591 [==============================] - 20s 971us/sample - los
s: 0.0771 - accuracy: 0.9943 - val_loss: 0.0567 - val_accuracy: 0.9997
Epoch 6/10
20591/20591 [==============================] - 20s 963us/sample - los
s: 0.0753 - accuracy: 0.9943 - val_loss: 0.0569 - val_accuracy: 0.9999
Epoch 7/10
20591/20591 [==============================] - 21s 1ms/sample - loss:
0.0714 - accuracy: 0.9942 - val_loss: 0.0577 - val_accuracy: 0.9993
Epoch 8/10
20591/20591 [==============================] - 21s 1ms/sample - loss:
0.0584 - accuracy: 0.9977 - val_loss: 0.0465 - val_accuracy: 1.0000
Epoch 9/10
20591/20591 [==============================] - 19s 933us/sample - los
s: 0.0700 - accuracy: 0.9949 - val_loss: 0.0553 - val_accuracy: 0.9988
Epoch 10/10
20591/20591 [==============================] - 21s 1ms/sample - loss:
0.0521 - accuracy: 0.9985 - val_loss: 0.0387 - val_accuracy: 1.0000
7172/7172 - 2s - loss: 0.2741 - accuracy: 0.9435

Точность на проверочных данных: 0.9435304
```

In [5]:

```
def plot_history(histories, key='binary_crossentropy'):
    plt.figure(figsize=(16,10))

    for name, history in histories:
        val = plt.plot(history.epoch, history.history['val_' + key],
                       '--', label=name.title()+' Val')
        plt.plot(history.epoch, history.history[key], color=val[0].get_color(),
                 label=name.title()+' Train')

    plt.xlabel('Epochs')
    plt.ylabel(key.replace('_',' ').title())
    plt.legend()
    plt.xlim([0,max(history.epoch)])
    plt.show()

plot_history([('basic model', basic_model_history)], key='loss')
```



Задание 3. Примените дополнение данных (data augmentation). Как это повлияло на качество классификатора?

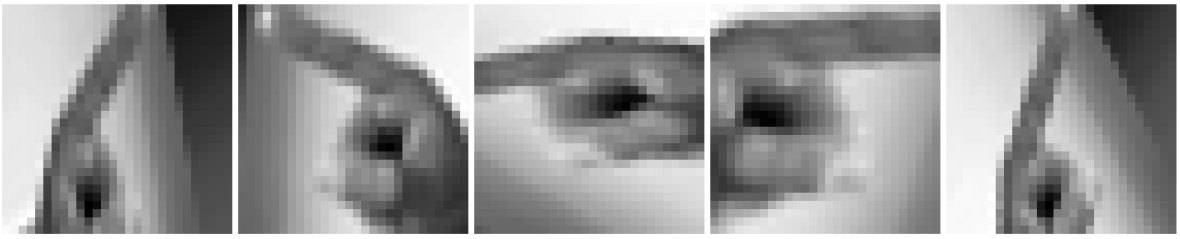
In [6]:

```
aug_train_image_gen = ImageDataGenerator(
    rotation_range=45,
    width_shift_range=.15,
    height_shift_range=.15,
    horizontal_flip=True,
    zoom_range=0.5
).flow(train_images, train_labels, batch_size=64, shuffle=True)

aug_valid_image_gen = ImageDataGenerator(
    rotation_range=45,
    width_shift_range=.15,
    height_shift_range=.15,
    horizontal_flip=True,
    zoom_range=0.5
).flow(valid_images, valid_labels, batch_size=64, shuffle=True)

aug_test_image_gen = ImageDataGenerator(
    rotation_range=45,
    width_shift_range=.15,
    height_shift_range=.15,
    horizontal_flip=True,
    zoom_range=0.5
).flow(test_images, test_labels, batch_size=64, shuffle=True)
```

In [7]:

```
plot_samples([aug_train_image_gen[0][0][0] for i in range(5)], 5, 'augmented')
```

augmented



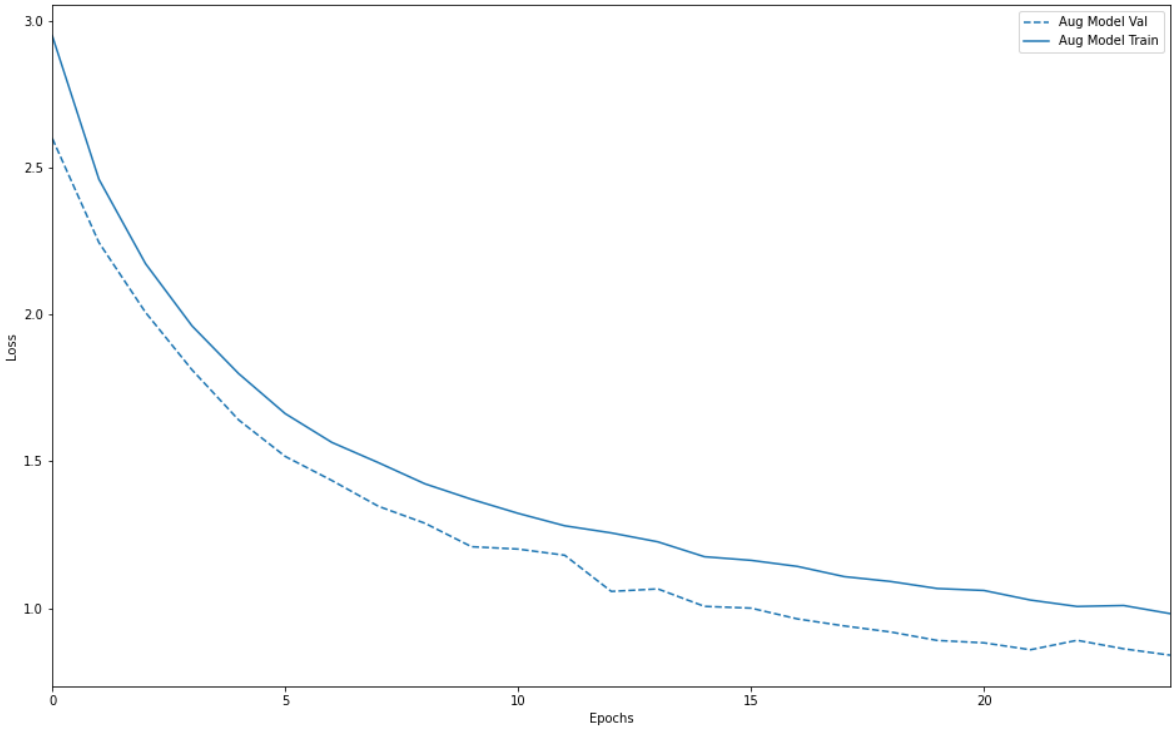
In [8]:

```
l2_regularization = 1e-4
num_classes = max(classes) + 1

aug_model = keras.Sequential([
    keras.layers.Conv2D(32, 3, activation='relu', padding='same', input_shape=(28,
    keras.layers.MaxPooling2D(pool_size=(2, 2)),

    keras.layers.Conv2D(64, 3, activation='relu', padding='same'),
    keras.layers.MaxPooling2D(pool_size=(2, 2)),

    keras.layers.Flatten(),
    keras.layers.Dense(units=512, activation='relu', kernel_regularizer=regularizer
    keras.layers.Dropout(0.25),

    keras.layers.Dense(units=num_classes, activation = 'softmax')
])

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

aug_model.summary()
aug_model_history = aug_model.fit(aug_train_image_gen, epochs=25, validation_data=a
test_loss, test_acc = aug_model.evaluate(aug_test_image_gen, verbose=2)

print('\nТочность на проверочных данных:', test_acc)
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 14, 14, 64)        18496
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 3136)              0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               1606144
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 25)                12825
```

In [9]:

```
plot_history([('aug model', aug_model_history)], key='loss')
```



Задание 4. Поэкспериментируйте с готовыми нейронными сетями (например, AlexNet, VGG16, Inception и т.п.), применив передаточное обучение. Как это повлияло на качество классификатора? Можно ли было обойтись без него?

In [15]:

```
pre_trained_model = keras.applications.VGG19(input_shape=(32, 32, 3), include_top=F

for i, layer in enumerate(pre_trained_model.layers):
    if i <= 10:
        layer.trainable = False
    else:
        layer.trainable = True

last_layer = pre_trained_model.get_layer('block5_pool')
last_output = last_layer.output

model = keras.layers.GlobalAveragePooling2D()(last_output)
model = keras.layers.Dense(512, activation='relu')(model)
model = keras.layers.Dropout(0.5)(model)
model = keras.layers.Dense(25, activation='softmax')(model)

vgg_model = keras.models.Model(pre_trained_model.input, model)

vgg_model.compile(loss='sparse_categorical_crossentropy',
                  optimizer=keras.optimizers.SGD(lr=1e-4, momentum=0.9),
                  metrics=['accuracy'])

vgg_model.summary()
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 32, 32, 3)]       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 32, 32, 64)        1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 32, 32, 64)        36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 16, 16, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 16, 16, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 16, 16, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 8, 8, 128)         0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 8, 8, 256)         295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 8, 8, 256)         590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 8, 8, 256)         590080
_________________________________________________________________
block3_conv4 (Conv2D)        (None, 8, 8, 256)         590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 4, 4, 256)         0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 4, 4, 512)         1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
```

```
block4_conv4 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 2, 2, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 2, 2, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 2, 2, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 2, 2, 512)         2359808
_________________________________________________________________
block5_conv4 (Conv2D)        (None, 2, 2, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 1, 1, 512)         0
_________________________________________________________________
global_average_pooling2d_1 ( (None, 512)               0
_________________________________________________________________
dense_6 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_7 (Dense)              (None, 25)                12825
=================================================================
Total params: 20,299,865
Trainable params: 17,974,297
Non-trainable params: 2,325,568
_________________________________________________________________
```

In [11]:

```
# форматируем данные, чтобы они подходили под параметры входа модели.

train_images_3d = np.array([np.repeat(img, 3, 2) for img in tf.image.resize(train_i
valid_images_3d = np.array([np.repeat(img, 3, 2) for img in tf.image.resize(valid_i
test_images_3d = np.array([np.repeat(img, 3, 2) for img in tf.image.resize(test_ima
```

In [20]:

```
vgg_model_history = vgg_model.fit(
    train_images_3d,
    train_labels,
    epochs=3,
    validation_data=(valid_images_3d, valid_labels))
```

```
Train on 20591 samples, validate on 6864 samples
Epoch 1/3
20591/20591 [==============================] - 1068s 52ms/sample - los
s: 0.3154 - accuracy: 0.8954 - val_loss: 0.0864 - val_accuracy: 0.9792
Epoch 2/3
20591/20591 [==============================] - 1112s 54ms/sample - los
s: 0.1063 - accuracy: 0.9678 - val_loss: 0.0190 - val_accuracy: 0.9977
Epoch 3/3
20591/20591 [==============================] - 1086s 53ms/sample - los
s: 0.0523 - accuracy: 0.9863 - val_loss: 0.0075 - val_accuracy: 0.9990
```

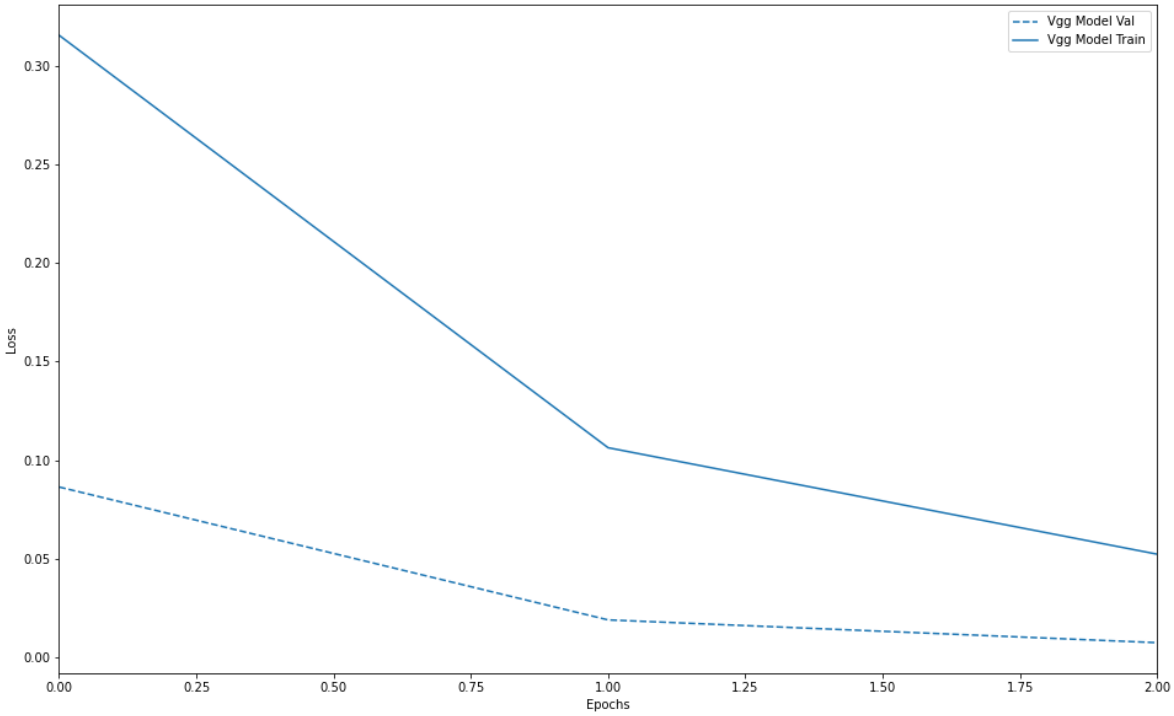
In [18]:

```
test_loss, test_acc = vgg_model.evaluate(test_images_3d, test_labels, verbose=2)

print('\nТочность на проверочных данных:', test_acc)
```

```
7172/7172 - 44s - loss: 0.4712 - accuracy: 0.8291

Точность на проверочных данных: 0.82905746
```

In [21]:

```
plot_history([('vgg model', vgg_model_history)], key='loss')
```



In [ ]: