Best Artworks Of All Time

Roman Gorb,
gorb.roman1999@gmail.com

In this work I will try to solve artist classification problem based on **"Best Artworks Of All Time"** (https://www.kaggle.com/ikarus777/best-artworks-of-all-time) dataset using `TensorFlow 2.0` .

Dataset description

After being challenged many times by my girlfriend about who is the best to guess the painter, I decided to use the power of machine learning to defeat her. I gathered a collection of artworks of the 50 most influential artists of all time. I added a dataset with basic information retrieved from wikipedia. I planned to create a convolutional neural network to recognise the artists looking the colors used and the geometric patterns inside the pictures.

Content

This dataset contains three files:

- `artists.csv` : dataset of information for each artist
- `images.zip` : collection of images (full size), divided in folders and sequentially numbered
- `resized.zip` : same collection but images have been resized and extracted from folder structure

Use resized.zip allows you to download less data and process faster your model.

Inspiration

My goal is to create a model that learn to identify the artist analysing new pictures. I hope to learn new techniques from public kernels or see some interesting usage of this data.

In [18]:

```
# Imports
import datetime
import os
import pathlib
import random
import shutil

from conf import *
from stats import *

os.environ['CUDA_DEVICE_ORDER'] = 'PCI_BUS_ID'
os.environ['CUDA_VISIBLE_DEVICES'] = '6'

STORAGE_DIR = '/data/rvgorb/hw10'

import tensorflow as tf
from tensorflow.keras.layers import (Conv2D, Dense, Dropout, Flatten,
                                     MaxPooling2D, BatchNormalization)
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping

%load_ext tensorboard

AUTOTUNE = tf.data.experimental.AUTOTUNE
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

Loading and exploring the data

In [6]:

```
# Data loading and extracting

# !wget -O /data/rvgorb/hw10/data.tar.xz https://www.dropbox.com/s/w90m55pl7ylgi
af/data.tar.xz?dl=0
# !tar -xf /data/rvgorb/hw10/data.tar.xz data
!ls /data/rvgorb/hw10/data/train | wc -l
!find /data/rvgorb/hw10/data/train -type f | wc -l
```

```
50
4099
```

In `train` data we have 51 artists(labels) and 6116 paintings(samples).

Let's have a look on our data:

In [12]:

```
# Visualization
fig = plt.figure(figsize=(16, 9))
ax = fig.gca()
path_to_img = f'{STORAGE_DIR}/data/train/William_Turner/William_Turner_9.jpg'
image = plt.imread(path_to_img)
ax.grid(False)
ax.set_xticks([])
ax.set_yticks([])
plt.title('William Turner')
_ = plt.imshow(image)
```

William Turner

Processing `train/val` split:

In [8]:

```
# Train/val split
os.makedirs(f'{STORAGE_DIR}/data/val', exist_ok=True)

TRAIN_FRAC = 0.7

# reading dir's names
ARTIST_LIST = {i:name for i, name in enumerate(os.listdir(f'{STORAGE_DIR}/data/train/'))}
IMAGES_DIR = f'{STORAGE_DIR}/data/train/'

max_train_images = 0

for artist in ARTIST_LIST.values():
    # preparing dir for every artist
    os.makedirs(f'{STORAGE_DIR}/data/val/{artist}/', exist_ok=True)

    # loading paths
    artist_path = f'{IMAGES_DIR}/{artist}/'
    images_filename = os.listdir(artist_path)

    # processing split
    num_train = int(len(images_filename) * TRAIN_FRAC)
    max_train_images = max(max_train_images, num_train)
    val_images = images_filename[num_train:]

    print(f'{artist} | train images = {num_train} | val images = {len(val_images)}')

    # saving to directory
    for image_filename in val_images:
        source = f'{IMAGES_DIR}/{artist}/{image_filename}'
        destination = f'{STORAGE_DIR}/data/val/{artist}/{image_filename}'
        shutil.copy(source, destination)
        os.remove(source)
```

```
Kazimir_Malevich | train images = 61 | val images = 27
Henri_Rousseau | train images = 34 | val images = 15
Hieronymus_Bosch | train images = 66 | val images = 29
Joan_Miro | train images = 49 | val images = 22
Giotto_di_Bondone | train images = 58 | val images = 25
Michelangelo | train images = 23 | val images = 11
Paul_Cezanne | train images = 22 | val images = 10
Paul_Gauguin | train images = 151 | val images = 66
Andrei_Rublev | train images = 48 | val images = 21
Mikhail_Vrubel | train images = 83 | val images = 36
Eugene_Delacroix | train images = 14 | val images = 7
Claude_Monet | train images = 35 | val images = 16
Edouard_Manet | train images = 43 | val images = 19
Jackson_Pollock | train images = 11 | val images = 5
Marc_Chagall | train images = 116 | val images = 51
Francisco_Goya | train images = 142 | val images = 61
Camille_Pissarro | train images = 44 | val images = 19
Alfred_Sisley | train images = 126 | val images = 55
Titian | train images = 124 | val images = 54
Gustave_Courbet | train images = 28 | val images = 13
Henri_Matisse | train images = 91 | val images = 39
Vasiliy_Kandinskiy | train images = 42 | val images = 19
Georges_Seurat | train images = 21 | val images = 9
Edgar_Degas | train images = 343 | val images = 148
Diego_Velazquez | train images = 62 | val images = 27
Jan_van_Eyck | train images = 39 | val images = 17
Salvador_Dali | train images = 67 | val images = 30
Sandro_Botticelli | train images = 79 | val images = 35
Rembrandt | train images = 128 | val images = 55
Pierre-Auguste_Renoir | train images = 164 | val images = 71
Edvard_Munch | train images = 32 | val images = 14
Peter_Paul_Rubens | train images = 68 | val images = 30
Frida_Kahlo | train images = 58 | val images = 26
Albrecht_DuтXa├кrer | train images = 160 | val images = 69
El_Greco | train images = 42 | val images = 18
Andy_Warhol | train images = 88 | val images = 38
William_Turner | train images = 32 | val images = 14
Caravaggio | train images = 26 | val images = 12
Rene_Magritte | train images = 94 | val images = 41
Henri_de_Toulouse-Lautrec | train images = 39 | val images = 17
Gustav_Klimt | train images = 56 | val images = 25
Diego_Rivera | train images = 34 | val images = 15
Raphael | train images = 53 | val images = 23
Albrecht_Du╠Иrer | train images = 160 | val images = 69
Piet_Mondrian | train images = 40 | val images = 18
Paul_Klee | train images = 91 | val images = 40
Vincent_van_Gogh | train images = 429 | val images = 184
Pieter_Bruegel | train images = 65 | val images = 28
Leonardo_da_Vinci | train images = 70 | val images = 30
Amedeo_Modigliani | train images = 94 | val images = 41
Pablo_Picasso | train images = 214 | val images = 93
```

Note:

- As you can see, there are two copies of Albrecht Dürer directory, so we should remove one duplicate.

In [9]:

```
# Removing duplicates
!rm -rf /data/rvgorb/hw10/data/train/Albrecht_DuтXa├кrer/
!rm -rf /data/rvgorb/hw10/data/val/Albrecht_DuтXa├кrer/
!rm -rf /data/rvgorb/hw10/data/test/Albrecht_DuтXa├кrer/
del ARTIST_LIST[33]
```

In [10]:

```
# Sanity check
!ls /data/rvgorb/hw10/data/train | wc -l
!ls /data/rvgorb/hw10/data/val | wc -l
!ls /data/rvgorb/hw10/data/test | wc -l
```

```
50
50
50
```

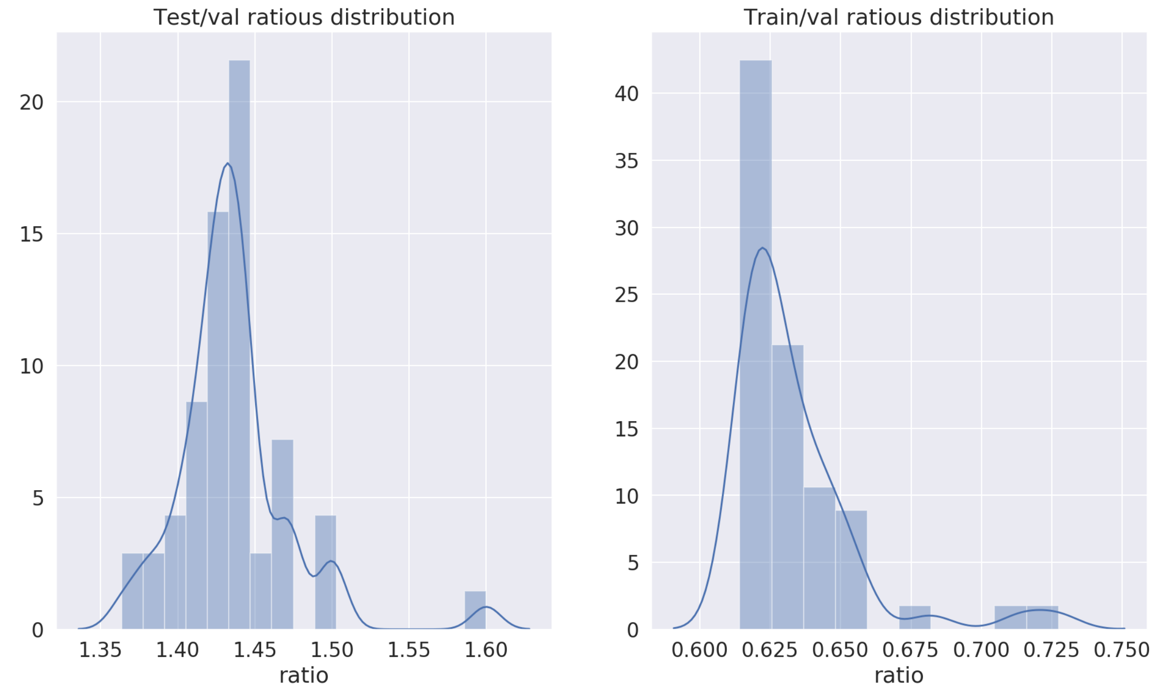
Checking that class balances on train, val and test data match:

In [11]:

```
# Ratious calculation
ratious_val = []
ratious_train = []

for artist in ARTIST_LIST.values():
    train_dir = f'{STORAGE_DIR}/data/train/{artist}'
    filenames = os.listdir(train_dir)
    num_train = len(filenames)

    test_dir = f'{STORAGE_DIR}/data/test/{artist}'
    filenames = os.listdir(test_dir)
    num_test = len(filenames)

    val_dir = f'{STORAGE_DIR}/data/val/{artist}'
    filenames = os.listdir(val_dir)
    num_val = len(filenames)

    ratious_val.append(num_test / num_val)
    ratious_train.append(num_test / num_train)

ratious_val = np.array(ratious_val)
ratious_train = np.array(ratious_train)
```

In [12]:

```
# Visualization
plt.figure(figsize=(16, 9))

plt.subplot(1, 2, 1)
sns.distplot(ratious_val)
plt.title('Test/val ratious distribution')
_ = plt.xlabel('ratio')

plt.subplot(1, 2, 2)
sns.distplot(ratious_train)
plt.title('Train/val ratious distribution')
_ = plt.xlabel('ratio')
```



Conclusions:

> - The ratious are mostly the same, that is why we can say that the splits are equally distributed.

This classification task is unbalanced, and there are some ways to deal with this problem:

- Random oversampling – creating copies of samples in minority classes.
- Random undersampling – drop samples from major classes.
- Do both to achieve the same number of pictures in each class.

You can find the method combining this ideas further. I tried to use this techniques, but they did not help me in my particular task.

But in this work I will mostly concentrate on the accuracy maximization, therefore there is no need to worry about the classes balance(which of course is not true when we consider e.g. F_1-score).

In [13]:

```
def under_over_sample(directory: str, level: int) -> None:
    filenames = os.listdir(directory)
    num_files = len(filenames)

    if num_files < level:
        # over sample
        num_add = level - num_files
        indexes_to_add = np.random.choice(num_files, size=num_add)
        for i, ind in enumerate(indexes_to_add):
            name = filenames[ind]
            source = f'{directory}/{name}'
            dest = f'{directory}/{name}_{i}'
            shutil.copy(source, dest)
    else:
        # under sample
        num_remove = num_files - level
        indexes_to_remove = np.random.choice(num_files, size=num_remove, replace
=False)
        for ind in indexes_to_remove:
            name = filenames[ind]
            file = f'{directory}/{name}'
            os.remove(file)

    filenames = os.listdir(directory)
    num_files = len(filenames)
    assert num_files == level, "WTF"  # sanity check
```

In [14]:

```
# Under/over sampling
# NUM_TRAIN = 200

# for artist in ARTIST_LIST.values():
#     train_dir = f'{STORAGE_DIR}/data/train/{artist}'
#     under_over_sample(train_dir, level=NUM_TRAIN)
```

Creating datasets

This part of work concetrates on network input data preparation.

In [31]:

```
# Paths config
train_dir = f'{STORAGE_DIR}/data/train/'
val_dir = f'{STORAGE_DIR}/data/val/'
test_dir = f'{STORAGE_DIR}/data/test/'
```

Implementing some custom augmentations(but I do not use them all in this work):

In [ ]:

```
# Custom augmentations
def pad_if_needed(img_tensor, sz=224):
    h, w = img_tensor.shape.as_list()[-3:-1]
    diff = tf.constant([sz, sz]) - tf.constant([h, w])

    need_pad = tf.math.maximum(diff, tf.zeros(2, dtype=tf.int32))
    need_pad = need_pad.numpy()

    pad_x = need_pad[0] // 2
    pad_y = need_pad[1] // 2

    return tf.pad(img_tensor,
                  tf.constant([[pad_x, need_pad[0] - pad_x],
                               [pad_y, need_pad[1] - pad_y], [0, 0]]),
                  mode='SYMMETRIC')


def center_crop(img_tensor, sz=224):
    assert sz % 2 == 0, 'sz should be even'

    h, w = img_tensor.shape.as_list()[-3:-1]
    c_h = h // 2
    c_w = w // 2

    return tf.image.crop_to_bounding_box(img_tensor, c_h - sz // 2,
                                         c_w - sz // 2, sz, sz)


def normalize(img_tensor, means=[0.485, 0.456, 0.406], stds=[0.229, 0.224, 0.225
]):
    mean = tf.constant(means, dtype=tf.float32)
    mean = tf.reshape(mean, [1, 1, 3])
    std = tf.constant(stds, dtype=tf.float32)
    std = tf.reshape(std, [1, 1, 3])

    img_tensor = img_tensor - mean
    img_tensor = img_tensor / std

    return img_tensor


def random_sized_random_crop(img_tensor, min_ratio=0.5, max_ratio=1.):
    h, w = img_tensor.shape.as_list()[-3:-1]

    min_h = int(h * min_ratio)
    max_h = int(h * max_ratio)
    min_w = int(w * min_ratio)
    max_w = int(w * max_ratio)

    new_h = tf.random.uniform([], minval=min_h, maxval=max_h, dtype=tf.int32)
    new_w = tf.random.uniform([], minval=min_w, maxval=max_w, dtype=tf.int32)

    return tf.image.random_crop(img_tensor, (new_h, new_w, 3))
```

Creating `train/test` preprocessing pipelines:

In [37]:

```
# Image preprocessing
def preprocess_train_image(image, sz=224):
    image = tf.image.decode_jpeg(image, channels=3)
    image = tf.image.convert_image_dtype(image, tf.float32)

    # zero center and normalize
    # image = tf.image.per_image_standardization(image)

    # custom normalization
    image = normalize(image)

    # handling small images
    # image = pad_if_needed(image)

    # horizontal flip
    image = tf.image.flip_left_right(image)

    # vertical flip
    # image = tf.image.flip_up_down(image)

    # brightness
    # image = tf.image.random_brightness(image, 0.5)

    # rotation
    # image = tf.image.rot90(image, tf.random.uniform(shape=[], minval=-1, maxva
l=1, dtype=tf.int32))

    # random crop to sz x sz
    # image = tf.image.random_crop(image, size=[sz, sz, 3])

    # small noise
    # noise = tf.random.normal(shape=tf.shape(image), mean=0.0, stddev=0.0001, d
type=tf.float32)
    # image = image + noise

    # random crop of random size
    image = random_sized_random_crop(image, min_ratio=0.75)

    # resize
    image = tf.image.resize(image, (sz, sz), method=tf.image.ResizeMethod.LANCZO
S5)

    return image


def preprocess_test_image(image, sz=224):
    image = tf.image.decode_jpeg(image, channels=3)
    image = tf.image.convert_image_dtype(image, tf.float32)

    # zero center and normalize
    # image = tf.image.per_image_standardization(image)

    # custom normalization
    image = normalize(image)

    # handling small images
    # image = pad_if_needed(image)

    # random crop to 224x224
```

```
    # image = center_crop(image)

    # resize
    image = tf.image.resize(image, (sz, sz), method=tf.image.ResizeMethod.LANCZO
S5)

    return image


def load_and_preprocess_train_image(path):
    image = tf.io.read_file(path)
    return preprocess_train_image(image)


def load_and_preprocess_test_image(path):
    image = tf.io.read_file(path)
    return preprocess_test_image(image)
```

Decorate them with tensorflow mappers:

In [38]:

```
# Mappers
train_mapper = lambda x: tf.py_function(
    func=load_and_preprocess_train_image, inp=[x], Tout=tf.float32)
test_mapper = lambda x: tf.py_function(
    func=load_and_preprocess_test_image, inp=[x], Tout=tf.float32)
```

Creating datasets:

In [39]:

```
def set_shapes(img, label, img_shape=(224, 224, 3)):
    img.set_shape(img_shape)
    label.set_shape([])
    return img, label


def create_dataset(dir_path, map_fn):
    root = pathlib.Path(dir_path)
    # Image paths parsing
    all_image_paths = list(root.glob('*/*'))
    all_image_paths = [str(path) for path in all_image_paths]
    random.shuffle(all_image_paths)
    # Labels creation
    label_names = sorted(item.name for item in root.glob('*/')
                         if item.is_dir())
    label_to_index = dict(
        (name, index) for index, name in enumerate(label_names))
    global index_to_label
    index_to_label = dict((index, name) for index, name in enumerate(label_names
))
    all_image_labels = [
        label_to_index[pathlib.Path(path).parent.name]
        for path in all_image_paths
    ]
    # Image dataset creation
    image_path_dataset = tf.data.Dataset.from_tensor_slices(all_image_paths)
    image_dataset = image_path_dataset.map(map_fn, num_parallel_calls=AUTOTUNE)
    # Labels dataset creation
    label_dataset = tf.data.Dataset.from_tensor_slices(
        tf.cast(all_image_labels, tf.int64))
    # Image + Label dataset creation
    image_label_dataset = tf.data.Dataset.zip((image_dataset, label_dataset))

    image_label_dataset = image_label_dataset.map(
        lambda img, lavel: set_shapes(img, lavel))

    return image_label_dataset, len(all_image_paths), len(label_names)
```

In [40]:

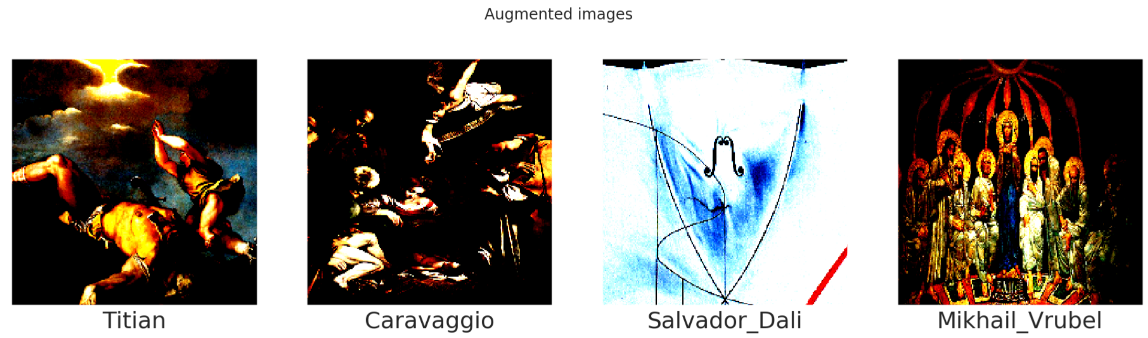
```
image_label_train, train_size, labels_count = create_dataset(train_dir, train_ma
pper)
image_label_val, val_size, _ = create_dataset(val_dir, test_mapper)
image_label_test, test_size, _ = create_dataset(test_dir, test_mapper)
```

Let's look at network input:

In [41]:

```
# Augmented images visualization
plt.figure(figsize=(16, 4))

for n, (image, label) in enumerate(image_label_train.take(4)):
    plt.subplot(1, 4, n + 1)
    plt.imshow(image)
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])
    plt.xlabel(index_to_label[label.numpy()])

_ = plt.suptitle('Augmented images')
```

```
Clipping input data to the valid range for imshow with RGB data
([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data
([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data
([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data
([0..1] for floats or [0..255] for integers).
```



Preparing datasets for efficient training and validation:

- Using cache to improve performance(storing in RAM).
- Shuffling train.
- Repeating datasets to avoid their exhaustibility.
- Using batching to fit in GPU memory.
- Using prefetch to load data in parallel.

In [42]:

```
TRAIN_BATCH_SIZE = 64

ds_train = image_label_train.cache()
ds_train = ds_train.shuffle(buffer_size=train_size)
ds_train = ds_train.repeat()
ds_train = ds_train.batch(TRAIN_BATCH_SIZE)
ds_train = ds_train.prefetch(buffer_size=AUTOTUNE)
```

In [43]:

```
VAL_BATCH_SIZE = 64

ds_val = image_label_val.cache()
ds_val = ds_val.repeat()
ds_val = ds_val.batch(VAL_BATCH_SIZE)
ds_val = ds_val.prefetch(buffer_size=AUTOTUNE)
```

In [49]:

```
TEST_BATCH_SIZE = 64

ds_test = image_label_test.cache()
ds_test = ds_test.batch(VAL_BATCH_SIZE)
ds_test = ds_test.prefetch(buffer_size=AUTOTUNE)
```

Conv net

I tried something **as simple as possible**: repeated thrice a conv block and use dense with dropouts thereafter.

The architecture below was found by trial and error. To be honest, I trained about 80 different models. You can find the report describing my story in the end of this section.

In [45]:

```
# Model architecture
model = Sequential([
    Conv2D(32,
           5,
           padding='same',
           activation='relu',
           strides=2,
           input_shape=(224, 224, 3)),
    BatchNormalization(),
    MaxPooling2D(strides=2, padding='same'),
    Conv2D(64, 3, padding='same', strides=2, activation='relu'),
    BatchNormalization(),
    MaxPooling2D(strides=2, padding='same'),
    Conv2D(128, 3, padding='same', strides=2, activation='relu'),
    BatchNormalization(),
    MaxPooling2D(strides=2, padding='same'),
    Flatten(),
    Dropout(0.5),
    Dense(1024, activation='relu'),
    Dropout(0.5),
    Dense(512, activation='relu'),
    # Dropout(0.35),
    Dense(labels_count, activation='softmax')
])
```

In [46]:

```
# Model compilation
reduce_lr = ReduceLROnPlateau(monitor='val_accuracy',
                              factor=0.5,
                              patience=7,
                              min_delta=0.01,
                              min_lr=1e-7,
                              verbose=1)

early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy',
                                                  patience=40,
                                                  min_delta=0.005,
                                                  verbose=1)

model.compile(optimizer=tf.keras.optimizers.Adam(lr=0.005),
              loss='sparse_categorical_crossentropy',
              metrics=["accuracy"])
```

In [47]:

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_3 (Conv2D)            (None, 112, 112, 32)      2432      
_________________________________________________________________
batch_normalization_3 (Batch (None, 112, 112, 32)      128       
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 56, 56, 32)        0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 28, 28, 64)        18496     
_________________________________________________________________
batch_normalization_4 (Batch (None, 28, 28, 64)        256       
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 14, 14, 64)        0         
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 7, 7, 128)         73856     
_________________________________________________________________
batch_normalization_5 (Batch (None, 7, 7, 128)         512       
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 4, 4, 128)         0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 2048)              0         
_________________________________________________________________
dropout_3 (Dropout)          (None, 2048)              0         
_________________________________________________________________
dense_3 (Dense)              (None, 1024)              2098176   
_________________________________________________________________
dropout_4 (Dropout)          (None, 1024)              0         
_________________________________________________________________
dense_4 (Dense)              (None, 512)               524800    
_________________________________________________________________
dense_5 (Dense)              (None, 50)                25650     
=================================================================
Total params: 2,744,306
Trainable params: 2,743,858
Non-trainable params: 448
_________________________________________________________________
```

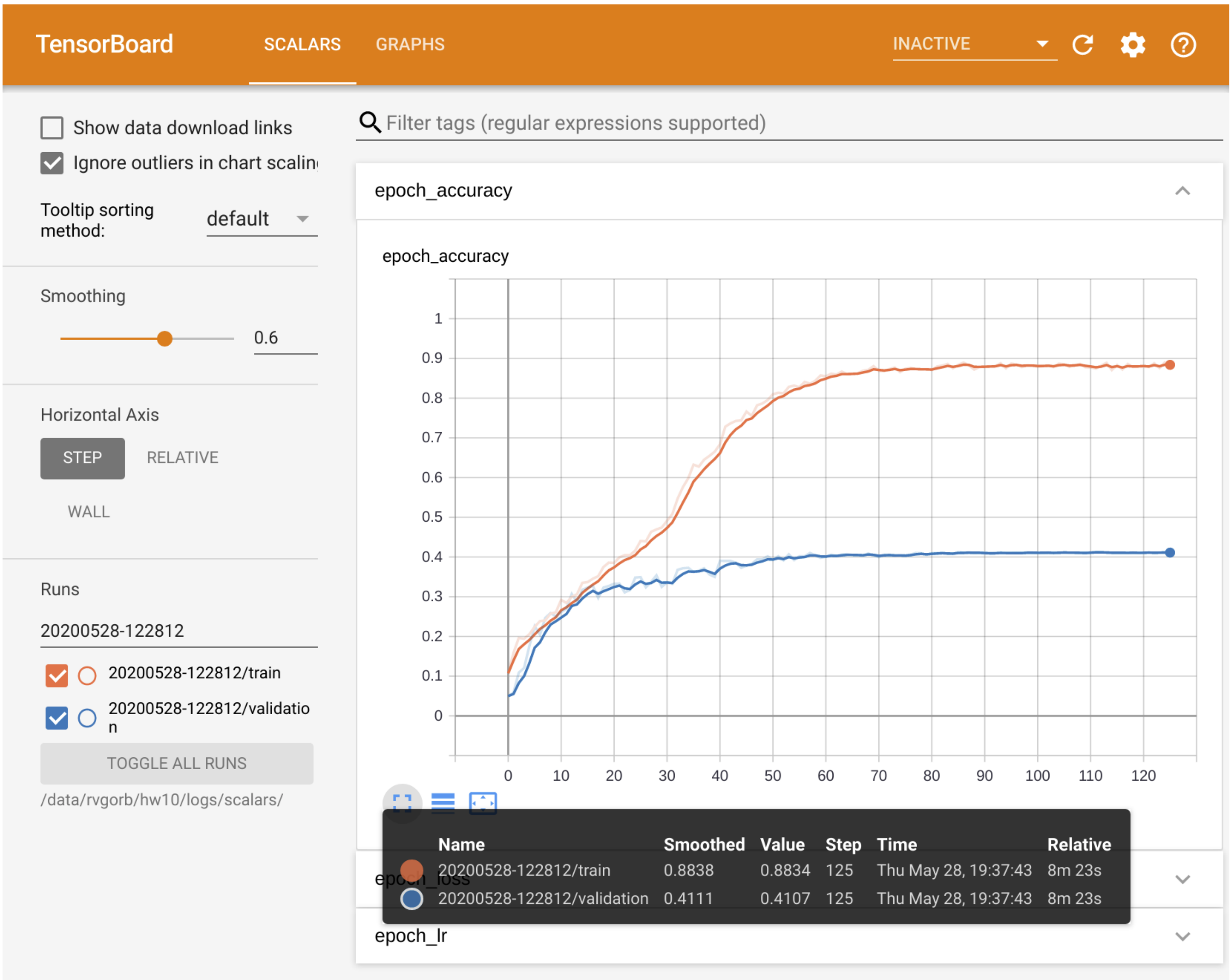
Using tensorboard to visualize training process:

In [27]:

```
tensorboard --logdir /data/rvgorb/hw10/logs/scalars/ --port 7775
```

```
Reusing TensorBoard on port 7775 (pid 5820), started 0:03:14 ago. (U
se '!kill 5820' to kill it.)
```

Saved results:


TensorBoard
SCALARS
GRAPHS
INACTIVE
Filter tags (regular expressions supported)
Show data download links
Ignore outliers in chart scalin
Tooltip sorting method:
default
Smoothing
0.6
Horizontal Axis
STEP
RELATIVE
WALL
Runs
20200528-122812
20200528-122812/train
20200528-122812/validation
TOGGLE ALL RUNS
/data/rvgorb/hw10/logs/scalars/
epoch_accuracy
epoch_accuracy
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
10
20
30
40
50
60
70
80
90
100
110
120
Name Smoothed Value Step Time Relative
20200528-122812/train 0.8838 0.8834 125 Thu May 28, 19:37:43 8m 23s
20200528-122812/validation 0.4111 0.4107 125 Thu May 28, 19:37:43 8m 23s
epoch_loss
epoch_lr

In [48]:

```
# My model fit
logdir = "/data/rvgorb/hw10/logs/scalars/" + datetime.datetime.now().strftime(
    "%Y%m%d-%H%M%S")
print("logdir:", logdir)
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir)

steps_per_epoch = int(tf.math.ceil(train_size / TRAIN_BATCH_SIZE).numpy())
validation_steps = int(tf.math.ceil(val_size / VAL_BATCH_SIZE).numpy())

model.fit(ds_train,
          epochs=300,
          validation_data=ds_val,
          steps_per_epoch=steps_per_epoch,
          validation_steps=validation_steps,
          callbacks=[reduce_lr, early_stopping, tensorboard_callback])
```

```
logdir: /data/rvgorb/hw10/logs/scalars/20200528-122812
Train for 65 steps, validate for 28 steps
Epoch 1/300
 1/65 [..............................] - ETA: 48:34 - loss: 6.0523 -
accuracy: 0.0312WARNING:tensorflow:Method (on_train_batch_end) is sl
ow compared to the batch update (0.360170). Check your callbacks.
65/65 [==============================] - 68s 1s/step - loss: 4.3549
- accuracy: 0.1070 - val_loss: 4.2660 - val_accuracy: 0.0497
Epoch 2/300
65/65 [==============================] - 5s 79ms/step - loss: 3.3285
- accuracy: 0.1587 - val_loss: 3.8013 - val_accuracy: 0.0586
Epoch 3/300
65/65 [==============================] - 4s 59ms/step - loss: 3.2114
- accuracy: 0.1962 - val_loss: 3.7575 - val_accuracy: 0.1088
Epoch 4/300
65/65 [==============================] - 4s 60ms/step - loss: 3.1242
- accuracy: 0.1942 - val_loss: 3.5532 - val_accuracy: 0.1205
Epoch 5/300
65/65 [==============================] - 5s 71ms/step - loss: 3.0687
- accuracy: 0.2055 - val_loss: 3.2310 - val_accuracy: 0.1758
Epoch 6/300
65/65 [==============================] - 4s 62ms/step - loss: 2.9782
- accuracy: 0.2243 - val_loss: 3.0044 - val_accuracy: 0.2254
Epoch 7/300
65/65 [==============================] - 5s 74ms/step - loss: 2.8929
- accuracy: 0.2385 - val_loss: 3.2009 - val_accuracy: 0.2042
Epoch 8/300
65/65 [==============================] - 4s 58ms/step - loss: 2.8416
- accuracy: 0.2406 - val_loss: 2.9233 - val_accuracy: 0.2467
Epoch 9/300
65/65 [==============================] - 4s 61ms/step - loss: 2.7955
- accuracy: 0.2575 - val_loss: 2.8215 - val_accuracy: 0.2589
Epoch 10/300
65/65 [==============================] - 4s 69ms/step - loss: 2.7435
- accuracy: 0.2608 - val_loss: 2.9356 - val_accuracy: 0.2506
Epoch 11/300
65/65 [==============================] - 4s 57ms/step - loss: 2.6722
- accuracy: 0.2921 - val_loss: 2.8232 - val_accuracy: 0.2612
Epoch 12/300
65/65 [==============================] - 4s 60ms/step - loss: 2.6694
- accuracy: 0.2844 - val_loss: 2.7801 - val_accuracy: 0.2701
Epoch 13/300
65/65 [==============================] - 5s 74ms/step - loss: 2.5713
- accuracy: 0.2998 - val_loss: 2.5888 - val_accuracy: 0.3069
Epoch 14/300
65/65 [==============================] - 4s 58ms/step - loss: 2.5278
- accuracy: 0.3063 - val_loss: 2.6960 - val_accuracy: 0.2868
Epoch 15/300
65/65 [==============================] - 4s 64ms/step - loss: 2.4653
- accuracy: 0.3349 - val_loss: 2.5527 - val_accuracy: 0.3175
Epoch 16/300
65/65 [==============================] - 4s 68ms/step - loss: 2.3853
- accuracy: 0.3368 - val_loss: 2.6066 - val_accuracy: 0.3231
Epoch 17/300
65/65 [==============================] - 4s 60ms/step - loss: 2.3822
- accuracy: 0.3454 - val_loss: 2.5492 - val_accuracy: 0.3270
Epoch 18/300
65/65 [==============================] - 4s 64ms/step - loss: 2.2962
- accuracy: 0.3517 - val_loss: 2.6232 - val_accuracy: 0.2969
Epoch 19/300
65/65 [==============================] - 4s 68ms/step - loss: 2.2563
```

```
- accuracy: 0.3750 - val_loss: 2.5733 - val_accuracy: 0.3231
Epoch 20/300
65/65 [==============================] - 4s 59ms/step - loss: 2.2117
- accuracy: 0.3858 - val_loss: 2.5241 - val_accuracy: 0.3270
Epoch 21/300
65/65 [==============================] - 4s 57ms/step - loss: 2.1697
- accuracy: 0.3851 - val_loss: 2.5634 - val_accuracy: 0.3326
Epoch 22/300
65/65 [==============================] - 4s 66ms/step - loss: 2.1131
- accuracy: 0.3990 - val_loss: 2.6421 - val_accuracy: 0.3331
Epoch 23/300
65/65 [==============================] - 4s 64ms/step - loss: 2.0992
- accuracy: 0.4046 - val_loss: 2.6743 - val_accuracy: 0.3108
Epoch 24/300
65/65 [==============================] - 4s 61ms/step - loss: 2.0906
- accuracy: 0.4046 - val_loss: 2.5791 - val_accuracy: 0.3170
Epoch 25/300
65/65 [==============================] - 4s 65ms/step - loss: 2.0266
- accuracy: 0.4163 - val_loss: 2.4872 - val_accuracy: 0.3482
Epoch 26/300
65/65 [==============================] - 4s 62ms/step - loss: 1.9657
- accuracy: 0.4404 - val_loss: 2.4858 - val_accuracy: 0.3493
Epoch 27/300
65/65 [==============================] - 4s 66ms/step - loss: 1.9472
- accuracy: 0.4394 - val_loss: 2.6680 - val_accuracy: 0.3237
Epoch 28/300
65/65 [==============================] - 4s 68ms/step - loss: 1.8955
- accuracy: 0.4635 - val_loss: 2.5711 - val_accuracy: 0.3376
Epoch 29/300
65/65 [==============================] - 5s 75ms/step - loss: 1.8382
- accuracy: 0.4695 - val_loss: 2.5757 - val_accuracy: 0.3538
Epoch 30/300
65/65 [==============================] - 4s 68ms/step - loss: 1.8183
- accuracy: 0.4743 - val_loss: 2.8030 - val_accuracy: 0.3248
Epoch 31/300
65/65 [==============================] - 4s 55ms/step - loss: 1.7819
- accuracy: 0.4911 - val_loss: 2.5744 - val_accuracy: 0.3365
Epoch 32/300
63/65 [============================>.] - ETA: 0s - loss: 1.7008 - ac
curacy: 0.5084
Epoch 00032: ReduceLROnPlateau reducing learning rate to 0.002499999
9441206455.
65/65 [==============================] - 4s 55ms/step - loss: 1.7022
- accuracy: 0.5082 - val_loss: 2.6395 - val_accuracy: 0.3326
Epoch 33/300
65/65 [==============================] - 4s 65ms/step - loss: 1.5440
- accuracy: 0.5471 - val_loss: 2.5041 - val_accuracy: 0.3677
Epoch 34/300
65/65 [==============================] - 5s 71ms/step - loss: 1.4346
- accuracy: 0.5760 - val_loss: 2.5036 - val_accuracy: 0.3717
Epoch 35/300
65/65 [==============================] - 4s 58ms/step - loss: 1.3108
- accuracy: 0.5981 - val_loss: 2.5636 - val_accuracy: 0.3728
Epoch 36/300
65/65 [==============================] - 4s 67ms/step - loss: 1.2362
- accuracy: 0.6325 - val_loss: 2.6507 - val_accuracy: 0.3622
Epoch 37/300
65/65 [==============================] - 5s 73ms/step - loss: 1.2452
- accuracy: 0.6276 - val_loss: 2.5160 - val_accuracy: 0.3655
Epoch 38/300
65/65 [==============================] - 4s 55ms/step - loss: 1.1630
```

```
- accuracy: 0.6450 - val_loss: 2.6169 - val_accuracy: 0.3717
Epoch 39/300
65/65 [==============================] - 4s 65ms/step - loss: 1.1283
- accuracy: 0.6546 - val_loss: 2.6458 - val_accuracy: 0.3560
Epoch 40/300
64/65 [============================>.] - ETA: 0s - loss: 1.1099 - ac
curacy: 0.6646
Epoch 00040: ReduceLROnPlateau reducing learning rate to 0.001249999
9720603228.
65/65 [==============================] - 4s 67ms/step - loss: 1.1078
- accuracy: 0.6651 - val_loss: 2.7512 - val_accuracy: 0.3499
Epoch 41/300
65/65 [==============================] - 4s 55ms/step - loss: 1.0042
- accuracy: 0.6839 - val_loss: 2.5400 - val_accuracy: 0.3906
Epoch 42/300
65/65 [==============================] - 3s 53ms/step - loss: 0.8918
- accuracy: 0.7281 - val_loss: 2.6070 - val_accuracy: 0.3901
Epoch 43/300
65/65 [==============================] - 4s 65ms/step - loss: 0.8509
- accuracy: 0.7365 - val_loss: 2.6151 - val_accuracy: 0.3901
Epoch 44/300
65/65 [==============================] - 5s 83ms/step - loss: 0.8242
- accuracy: 0.7425 - val_loss: 2.6806 - val_accuracy: 0.3856
Epoch 45/300
65/65 [==============================] - 4s 62ms/step - loss: 0.8143
- accuracy: 0.7440 - val_loss: 2.7091 - val_accuracy: 0.3728
Epoch 46/300
65/65 [==============================] - 5s 76ms/step - loss: 0.7602
- accuracy: 0.7661 - val_loss: 2.7830 - val_accuracy: 0.3783
Epoch 47/300
65/65 [==============================] - 5s 70ms/step - loss: 0.7694
- accuracy: 0.7553 - val_loss: 2.7791 - val_accuracy: 0.3828
Epoch 48/300
64/65 [============================>.] - ETA: 0s - loss: 0.6879 - ac
curacy: 0.7822
Epoch 00048: ReduceLROnPlateau reducing learning rate to 0.000624999
9860301614.
65/65 [==============================] - 5s 71ms/step - loss: 0.6894
- accuracy: 0.7820 - val_loss: 2.7264 - val_accuracy: 0.3940
Epoch 49/300
65/65 [==============================] - 4s 65ms/step - loss: 0.6546
- accuracy: 0.7865 - val_loss: 2.7858 - val_accuracy: 0.3940
Epoch 50/300
65/65 [==============================] - 4s 54ms/step - loss: 0.6377
- accuracy: 0.7952 - val_loss: 2.8016 - val_accuracy: 0.4018
Epoch 51/300
65/65 [==============================] - 4s 63ms/step - loss: 0.5974
- accuracy: 0.8099 - val_loss: 2.7655 - val_accuracy: 0.3929
Epoch 52/300
65/65 [==============================] - 5s 77ms/step - loss: 0.5653
- accuracy: 0.8135 - val_loss: 2.7885 - val_accuracy: 0.4018
Epoch 53/300
65/65 [==============================] - 4s 55ms/step - loss: 0.5878
- accuracy: 0.8120 - val_loss: 2.7982 - val_accuracy: 0.3917
Epoch 54/300
65/65 [==============================] - 4s 64ms/step - loss: 0.5405
- accuracy: 0.8284 - val_loss: 2.8237 - val_accuracy: 0.4068
Epoch 55/300
65/65 [==============================] - 4s 68ms/step - loss: 0.5271
- accuracy: 0.8315 - val_loss: 2.9271 - val_accuracy: 0.3923
Epoch 56/300
```

```
65/65 [==============================] - 4s 63ms/step - loss: 0.5246
- accuracy: 0.8274 - val_loss: 2.8571 - val_accuracy: 0.4018
Epoch 57/300
64/65 [============================>.] - ETA: 0s - loss: 0.5107 - ac
curacy: 0.8398
Epoch 00057: ReduceLROnPlateau reducing learning rate to 0.000312499
9930150807.
65/65 [==============================] - 4s 62ms/step - loss: 0.5074
- accuracy: 0.8409 - val_loss: 2.9077 - val_accuracy: 0.4018
Epoch 58/300
65/65 [==============================] - 4s 67ms/step - loss: 0.5037
- accuracy: 0.8363 - val_loss: 2.8205 - val_accuracy: 0.4102
Epoch 59/300
65/65 [==============================] - 4s 60ms/step - loss: 0.4964
- accuracy: 0.8438 - val_loss: 2.8777 - val_accuracy: 0.4040
Epoch 60/300
65/65 [==============================] - 4s 63ms/step - loss: 0.4459
- accuracy: 0.8572 - val_loss: 2.9136 - val_accuracy: 0.3979
Epoch 61/300
65/65 [==============================] - 5s 70ms/step - loss: 0.4516
- accuracy: 0.8548 - val_loss: 2.9007 - val_accuracy: 0.4007
Epoch 62/300
65/65 [==============================] - 4s 60ms/step - loss: 0.4361
- accuracy: 0.8615 - val_loss: 2.9478 - val_accuracy: 0.4057
Epoch 63/300
65/65 [==============================] - 4s 61ms/step - loss: 0.4399
- accuracy: 0.8594 - val_loss: 2.9295 - val_accuracy: 0.4023
Epoch 64/300
64/65 [============================>.] - ETA: 0s - loss: 0.4170 - ac
curacy: 0.8669
Epoch 00064: ReduceLROnPlateau reducing learning rate to 0.000156249
99650754035.
65/65 [==============================] - 5s 69ms/step - loss: 0.4193
- accuracy: 0.8668 - val_loss: 2.9794 - val_accuracy: 0.4062
Epoch 65/300
65/65 [==============================] - 4s 56ms/step - loss: 0.4382
- accuracy: 0.8606 - val_loss: 2.9342 - val_accuracy: 0.4074
Epoch 66/300
65/65 [==============================] - 4s 55ms/step - loss: 0.4376
- accuracy: 0.8618 - val_loss: 2.9305 - val_accuracy: 0.4051
Epoch 67/300
65/65 [==============================] - 4s 63ms/step - loss: 0.4178
- accuracy: 0.8625 - val_loss: 2.9316 - val_accuracy: 0.4046
Epoch 68/300
65/65 [==============================] - 4s 64ms/step - loss: 0.3973
- accuracy: 0.8690 - val_loss: 2.9394 - val_accuracy: 0.4035
Epoch 69/300
65/65 [==============================] - 4s 54ms/step - loss: 0.4161
- accuracy: 0.8712 - val_loss: 2.9180 - val_accuracy: 0.4090
Epoch 70/300
65/65 [==============================] - 3s 53ms/step - loss: 0.4106
- accuracy: 0.8805 - val_loss: 2.9361 - val_accuracy: 0.4040
Epoch 71/300
64/65 [============================>.] - ETA: 0s - loss: 0.4171 - ac
curacy: 0.8684
Epoch 00071: ReduceLROnPlateau reducing learning rate to 7.812499825
377017e-05.
65/65 [==============================] - 4s 61ms/step - loss: 0.4152
- accuracy: 0.8685 - val_loss: 2.9428 - val_accuracy: 0.3996
Epoch 72/300
65/65 [==============================] - 5s 70ms/step - loss: 0.4133
```

```
- accuracy: 0.8678 - val_loss: 2.9351 - val_accuracy: 0.4057
Epoch 73/300
65/65 [==============================] - 4s 60ms/step - loss: 0.4128
- accuracy: 0.8731 - val_loss: 2.9382 - val_accuracy: 0.4057
Epoch 74/300
65/65 [==============================] - 4s 54ms/step - loss: 0.3913
- accuracy: 0.8769 - val_loss: 2.9426 - val_accuracy: 0.4029
Epoch 75/300
65/65 [==============================] - 3s 53ms/step - loss: 0.4095
- accuracy: 0.8656 - val_loss: 2.9357 - val_accuracy: 0.4040
Epoch 76/300
65/65 [==============================] - 4s 67ms/step - loss: 0.3963
- accuracy: 0.8738 - val_loss: 2.9567 - val_accuracy: 0.4029
Epoch 77/300
65/65 [==============================] - 4s 69ms/step - loss: 0.3889
- accuracy: 0.8757 - val_loss: 2.9464 - val_accuracy: 0.4062
Epoch 78/300
63/65 [============================>.] - ETA: 0s - loss: 0.3986 - ac
curacy: 0.8720
Epoch 00078: ReduceLROnPlateau reducing learning rate to 3.906249912
6885086e-05.
65/65 [==============================] - 3s 53ms/step - loss: 0.3945
- accuracy: 0.8731 - val_loss: 2.9498 - val_accuracy: 0.4102
Epoch 79/300
65/65 [==============================] - 3s 53ms/step - loss: 0.3971
- accuracy: 0.8719 - val_loss: 2.9545 - val_accuracy: 0.4096
Epoch 80/300
65/65 [==============================] - 4s 65ms/step - loss: 0.3920
- accuracy: 0.8724 - val_loss: 2.9585 - val_accuracy: 0.4051
Epoch 81/300
65/65 [==============================] - 4s 63ms/step - loss: 0.3934
- accuracy: 0.8709 - val_loss: 2.9519 - val_accuracy: 0.4096
Epoch 82/300
65/65 [==============================] - 4s 56ms/step - loss: 0.3659
- accuracy: 0.8798 - val_loss: 2.9631 - val_accuracy: 0.4102
Epoch 83/300
65/65 [==============================] - 4s 54ms/step - loss: 0.3775
- accuracy: 0.8820 - val_loss: 2.9582 - val_accuracy: 0.4107
Epoch 84/300
65/65 [==============================] - 4s 62ms/step - loss: 0.3728
- accuracy: 0.8861 - val_loss: 2.9661 - val_accuracy: 0.4079
Epoch 85/300
64/65 [============================>.] - ETA: 0s - loss: 0.3882 - ac
curacy: 0.8755
Epoch 00085: ReduceLROnPlateau reducing learning rate to 1.953124956
3442543e-05.
65/65 [==============================] - 4s 69ms/step - loss: 0.3890
- accuracy: 0.8755 - val_loss: 2.9553 - val_accuracy: 0.4079
Epoch 86/300
65/65 [==============================] - 4s 57ms/step - loss: 0.3637
- accuracy: 0.8839 - val_loss: 2.9610 - val_accuracy: 0.4124
Epoch 87/300
65/65 [==============================] - 4s 54ms/step - loss: 0.3533
- accuracy: 0.8894 - val_loss: 2.9673 - val_accuracy: 0.4107
Epoch 88/300
65/65 [==============================] - 4s 61ms/step - loss: 0.3743
- accuracy: 0.8827 - val_loss: 2.9607 - val_accuracy: 0.4118
Epoch 89/300
65/65 [==============================] - 4s 66ms/step - loss: 0.3848
- accuracy: 0.8719 - val_loss: 2.9695 - val_accuracy: 0.4102
Epoch 90/300
```

```
65/65 [==============================] - 4s 59ms/step - loss: 0.3795
- accuracy: 0.8784 - val_loss: 2.9643 - val_accuracy: 0.4107
Epoch 91/300
65/65 [==============================] - 4s 54ms/step - loss: 0.3831
- accuracy: 0.8796 - val_loss: 2.9653 - val_accuracy: 0.4090
Epoch 92/300
65/65 [==============================] - 4s 58ms/step - loss: 0.3860
- accuracy: 0.8805 - val_loss: 2.9674 - val_accuracy: 0.4085
Epoch 93/300
63/65 [============================>.] - ETA: 0s - loss: 0.3705 - ac
curacy: 0.8802
Epoch 00093: ReduceLROnPlateau reducing learning rate to 9.765624781
721272e-06.
65/65 [==============================] - 5s 70ms/step - loss: 0.3705
- accuracy: 0.8808 - val_loss: 2.9630 - val_accuracy: 0.4096
Epoch 94/300
65/65 [==============================] - 4s 58ms/step - loss: 0.3679
- accuracy: 0.8877 - val_loss: 2.9646 - val_accuracy: 0.4113
Epoch 95/300
65/65 [==============================] - 4s 56ms/step - loss: 0.3925
- accuracy: 0.8752 - val_loss: 2.9754 - val_accuracy: 0.4096
Epoch 96/300
65/65 [==============================] - 5s 71ms/step - loss: 0.3728
- accuracy: 0.8870 - val_loss: 2.9687 - val_accuracy: 0.4096
Epoch 97/300
65/65 [==============================] - 4s 62ms/step - loss: 0.3708
- accuracy: 0.8846 - val_loss: 2.9663 - val_accuracy: 0.4102
Epoch 98/300
65/65 [==============================] - 3s 54ms/step - loss: 0.3872
- accuracy: 0.8784 - val_loss: 2.9718 - val_accuracy: 0.4096
Epoch 99/300
65/65 [==============================] - 4s 61ms/step - loss: 0.3860
- accuracy: 0.8829 - val_loss: 2.9739 - val_accuracy: 0.4118
Epoch 100/300
63/65 [============================>.] - ETA: 0s - loss: 0.3722 - ac
curacy: 0.8807
Epoch 00100: ReduceLROnPlateau reducing learning rate to 4.882812390
860636e-06.
65/65 [==============================] - 4s 66ms/step - loss: 0.3710
- accuracy: 0.8820 - val_loss: 2.9647 - val_accuracy: 0.4113
Epoch 101/300
65/65 [==============================] - 4s 55ms/step - loss: 0.3729
- accuracy: 0.8793 - val_loss: 2.9748 - val_accuracy: 0.4102
Epoch 102/300
65/65 [==============================] - 4s 54ms/step - loss: 0.3669
- accuracy: 0.8808 - val_loss: 2.9637 - val_accuracy: 0.4107
Epoch 103/300
65/65 [==============================] - 4s 60ms/step - loss: 0.3653
- accuracy: 0.8825 - val_loss: 2.9654 - val_accuracy: 0.4113
Epoch 104/300
65/65 [==============================] - 4s 65ms/step - loss: 0.3752
- accuracy: 0.8784 - val_loss: 2.9748 - val_accuracy: 0.4096
Epoch 105/300
65/65 [==============================] - 4s 57ms/step - loss: 0.3552
- accuracy: 0.8868 - val_loss: 2.9645 - val_accuracy: 0.4113
Epoch 106/300
65/65 [==============================] - 3s 53ms/step - loss: 0.3656
- accuracy: 0.8853 - val_loss: 2.9610 - val_accuracy: 0.4118
Epoch 107/300
63/65 [============================>.] - ETA: 0s - loss: 0.3825 - ac
curacy: 0.8785
```

```
Epoch 00107: ReduceLROnPlateau reducing learning rate to 2.441406195
430318e-06.
65/65 [==============================] - 4s 55ms/step - loss: 0.3800
- accuracy: 0.8788 - val_loss: 2.9635 - val_accuracy: 0.4107
Epoch 108/300
65/65 [==============================] - 4s 56ms/step - loss: 0.3733
- accuracy: 0.8832 - val_loss: 2.9651 - val_accuracy: 0.4107
Epoch 109/300
65/65 [==============================] - 4s 69ms/step - loss: 0.3808
- accuracy: 0.8861 - val_loss: 2.9705 - val_accuracy: 0.4102
Epoch 110/300
65/65 [==============================] - 4s 64ms/step - loss: 0.3968
- accuracy: 0.8779 - val_loss: 2.9689 - val_accuracy: 0.4096
Epoch 111/300
65/65 [==============================] - 4s 56ms/step - loss: 0.3796
- accuracy: 0.8774 - val_loss: 2.9642 - val_accuracy: 0.4118
Epoch 112/300
65/65 [==============================] - 4s 55ms/step - loss: 0.3778
- accuracy: 0.8745 - val_loss: 2.9645 - val_accuracy: 0.4129
Epoch 113/300
65/65 [==============================] - 4s 56ms/step - loss: 0.3563
- accuracy: 0.8820 - val_loss: 2.9680 - val_accuracy: 0.4118
Epoch 114/300
64/65 [============================>.] - ETA: 0s - loss: 0.3592 - ac
curacy: 0.8882
Epoch 00114: ReduceLROnPlateau reducing learning rate to 1.220703097
715159e-06.
65/65 [==============================] - 4s 67ms/step - loss: 0.3575
- accuracy: 0.8889 - val_loss: 2.9652 - val_accuracy: 0.4107
Epoch 115/300
65/65 [==============================] - 4s 68ms/step - loss: 0.3973
- accuracy: 0.8707 - val_loss: 2.9694 - val_accuracy: 0.4102
Epoch 116/300
65/65 [==============================] - 3s 54ms/step - loss: 0.3589
- accuracy: 0.8837 - val_loss: 2.9639 - val_accuracy: 0.4107
Epoch 117/300
65/65 [==============================] - 4s 60ms/step - loss: 0.3806
- accuracy: 0.8721 - val_loss: 2.9689 - val_accuracy: 0.4107
Epoch 118/300
65/65 [==============================] - 4s 63ms/step - loss: 0.3518
- accuracy: 0.8861 - val_loss: 2.9670 - val_accuracy: 0.4102
Epoch 119/300
65/65 [==============================] - 4s 55ms/step - loss: 0.3791
- accuracy: 0.8786 - val_loss: 2.9663 - val_accuracy: 0.4113
Epoch 120/300
65/65 [==============================] - 4s 54ms/step - loss: 0.3698
- accuracy: 0.8803 - val_loss: 2.9737 - val_accuracy: 0.4102
Epoch 121/300
63/65 [===========================>..] - ETA: 0s - loss: 0.3752 - ac
curacy: 0.8814
Epoch 00121: ReduceLROnPlateau reducing learning rate to 6.103515488
575795e-07.
65/65 [==============================] - 3s 54ms/step - loss: 0.3735
- accuracy: 0.8815 - val_loss: 2.9709 - val_accuracy: 0.4107
Epoch 122/300
65/65 [==============================] - 3s 52ms/step - loss: 0.3645
- accuracy: 0.8786 - val_loss: 2.9658 - val_accuracy: 0.4118
Epoch 123/300
65/65 [==============================] - 4s 56ms/step - loss: 0.3561
- accuracy: 0.8877 - val_loss: 2.9707 - val_accuracy: 0.4102
Epoch 124/300
```

```
65/65 [==============================] - 4s 63ms/step - loss: 0.3788
- accuracy: 0.8760 - val_loss: 2.9640 - val_accuracy: 0.4113
Epoch 125/300
65/65 [==============================] - 4s 60ms/step - loss: 0.3600
- accuracy: 0.8899 - val_loss: 2.9645 - val_accuracy: 0.4118
Epoch 126/300
65/65 [==============================] - 4s 55ms/step - loss: 0.3563
- accuracy: 0.8834 - val_loss: 2.9684 - val_accuracy: 0.4107
Epoch 00126: early stopping
```

Out[48]:

```
<tensorflow.python.keras.callbacks.History at 0x7fc782aa3be0>
```

Calculating test quality:

In [50]:

```
test_accuracy = model.evaluate(ds_test)[1]
```

```
40/40 [==============================] - 25s 631ms/step - loss: 3.06
80 - accuracy: 0.4017
```

Results:

In [16]:

```
def print_result(test_accuracy):
    print("Test result:")
    print("  test accuracy:\t\t{:.2f} %".format(
        test_accuracy * 100))

    if test_accuracy * 100 > 40:
        print("Achievement unlocked: ResNet152!")
    elif test_accuracy * 100 > 35:
        print("Achievement unlocked: Inception V3!")
    elif test_accuracy * 100 > 30:
        print("Achievement unlocked: AlexNet!")
    elif test_accuracy * 100 > 25:
        print("Achievement unlocked: MLP!")
    else:
        print("We need more \"layers\"! Follow instructons below")
```

In [17]:

```
print_result(test_accuracy)
```

```
Test result:
  test accuracy:                40.17 %
Achievement unlocked: ResNet152!
```

In [53]:

```
# Saving model
model.save('/data/rvgorb/hw10/my-final')
```

```
INFO:tensorflow:Assets written to: /data/rvgorb/hw10/my-final/assets
```

Report:

> It all started with conv blocks, I mean:
>
> ```
> - `Conv2d`
> - `ReLU`
> - `BatchNorm`
> - `MaxPooling2d`
> ```
>
> Of course, after that I added dense layers with activations.
>
> From the very beginning to almost the end I used `RandomCrop(224, 224)` (along with others), because it seemed incredibly logical to be. Let me explain. Many artists stand out in their style details: unique strokes, paint technique(e.g. impressionism), color tones and so on. Anyway we have to reduct image dimensionality to feed it into the network. So, if we want to save minute details of the painting we have to use only crops not rescaling. But to my surprise, I did not managed to achieve high score with this model.
>
> After several hours, I decided to replace crop with resize and badum-ts! I got got significant improvement on validation and test. But still the main problem I fought with was the overfitting: models was getting $\approx 100\%$ on `train-e` and only $\approx 37\%$ on validation.
>
> Another main improvement was to use `RandomCrop` to select some large subpart of the image(not 224×224). It brought significant variety to training data and reduced overfitting.
>
> I have done about 80 fits of the models to(in chronological order):
>
> ```
> - pick up conv blocks parameters to achieve overfitting
> - reduce overfitting
> ```
>
> I applied augmentations, but the models were **not training** after that:
>
> ```
> - per image standartization
> - brightness adjust
> - adding small gaussian noise
> ```
>
> I added dropouts and used large size crop and rescaling and after that I got current results.
>
> After two days of work, I got this results(model `20200528-122812`):
>
> - accuracy on training: 88.3%
> - accuracy on validation: 41%
> - accuracy on test: 40.2%

Transfer Learning

Now, I want to use pretrained on imagenet `ResNet50` :

In [77]:

```
# Model loading
resnet_base = tf.keras.applications.ResNet50V2(input_shape=(224, 224, 3),
                                                include_top=False,
                                                weights='imagenet')

resnet_base.trainable = True

from_layer = 187 # resnet-50 has 50 blocks, not layers

for layer in resnet_base.layers[:from_layer]:
    layer.trainable =  False
```

In [78]:

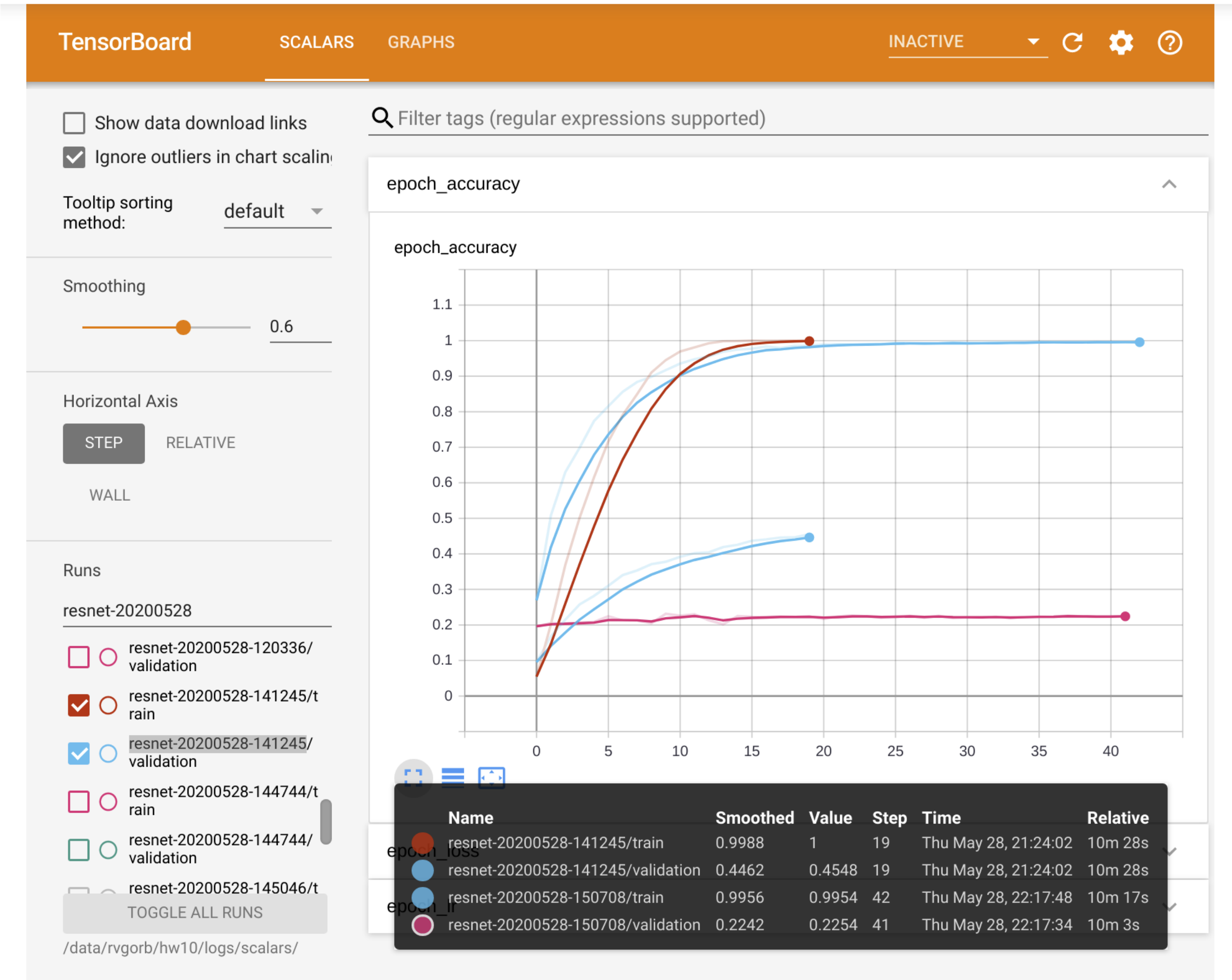
```
class Wrapper(tf.keras.Model):
    def __init__(self, base_model):
        super(Wrapper, self).__init__()

        self.base_model = base_model
        self.average_pooling_layer = tf.keras.layers.GlobalAveragePooling2D()
        self.output_layer = tf.keras.layers.Dense(50, activation='softmax')

    def call(self, inputs):
        x = self.base_model(inputs)
        x = self.average_pooling_layer(x)
        output = self.output_layer(x)
        return output
```

In [79]:

```
# Model compilation
base_learning_rate = 1e-5

resnet = Wrapper(resnet_base)
resnet.compile(optimizer=tf.keras.optimizers.RMSprop(lr=base_learning_rate),
               loss='sparse_categorical_crossentropy',
               metrics=['accuracy'])
```

In [140]:

```
tensorboard --logdir /data/rvgorb/hw10/logs/scalars/ --port 7775
```


TensorBoard
SCALARS
GRAPHS
INACTIVE
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method:
default
Smoothing
0.6
Horizontal Axis
STEP
RELATIVE
WALL
Runs
resnet-20200528
resnet-20200528-120336/validation
resnet-20200528-141245/train
resnet-20200528-141245/validation
resnet-20200528-144744/train
resnet-20200528-144744/validation
resnet-20200528-145046/t
TOGGLE ALL RUNS
/data/rvgorb/hw10/logs/scalars/
Filter tags (regular expressions supported)
epoch_accuracy
epoch_accuracy
epoch_loss
epoch_lr
Name Smoothed Value Step Time Relative
resnet-20200528-141245/train 0.9988 1 19 Thu May 28, 21:24:02 10m 28s
resnet-20200528-141245/validation 0.4462 0.4548 19 Thu May 28, 21:24:02 10m 28s
resnet-20200528-150708/train 0.9956 0.9954 42 Thu May 28, 22:17:48 10m 17s
resnet-20200528-150708/validation 0.2242 0.2254 41 Thu May 28, 22:17:34 10m 3s

In [80]:

```
# Model training
logdir = "/data/rvgorb/hw10/logs/scalars/resnet-" + datetime.datetime.now(
).strftime("%Y%m%d-%H%M%S")
print('logdir:', logdir)
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir)

history = resnet.fit(
    ds_train,
    epochs=20,
    validation_data=ds_val,
    steps_per_epoch=steps_per_epoch,
    validation_steps=validation_steps,
    callbacks=[tensorboard_callback, early_stopping, reduce_lr])
```

```
logdir: /data/rvgorb/hw10/logs/scalars/resnet-20200528-150708
Train for 65 steps, validate for 28 steps
Epoch 1/300
65/65 [==============================] - 19s 285ms/step - loss: 2.92
20 - accuracy: 0.2695 - val_loss: 4.4859 - val_accuracy: 0.1964
Epoch 2/300
65/65 [==============================] - 15s 234ms/step - loss: 1.85
50 - accuracy: 0.5082 - val_loss: 5.3787 - val_accuracy: 0.2054
Epoch 3/300
65/65 [==============================] - 15s 232ms/step - loss: 1.39
80 - accuracy: 0.6298 - val_loss: 6.1948 - val_accuracy: 0.2042
Epoch 4/300
65/65 [==============================] - 14s 220ms/step - loss: 1.11
47 - accuracy: 0.6986 - val_loss: 6.6686 - val_accuracy: 0.2065
Epoch 5/300
65/65 [==============================] - 14s 219ms/step - loss: 0.89
79 - accuracy: 0.7736 - val_loss: 6.9854 - val_accuracy: 0.2087
Epoch 6/300
65/65 [==============================] - 15s 233ms/step - loss: 0.73
92 - accuracy: 0.8154 - val_loss: 7.3262 - val_accuracy: 0.2243
Epoch 7/300
65/65 [==============================] - 14s 220ms/step - loss: 0.61
73 - accuracy: 0.8560 - val_loss: 7.8379 - val_accuracy: 0.2132
Epoch 8/300
65/65 [==============================] - 14s 219ms/step - loss: 0.51
63 - accuracy: 0.8837 - val_loss: 7.8087 - val_accuracy: 0.2126
Epoch 9/300
65/65 [==============================] - 14s 219ms/step - loss: 0.45
06 - accuracy: 0.8983 - val_loss: 8.5761 - val_accuracy: 0.2048
Epoch 10/300
65/65 [==============================] - 15s 226ms/step - loss: 0.38
71 - accuracy: 0.9171 - val_loss: 8.3120 - val_accuracy: 0.2316
Epoch 11/300
65/65 [==============================] - 15s 232ms/step - loss: 0.31
49 - accuracy: 0.9356 - val_loss: 8.8454 - val_accuracy: 0.2260
Epoch 12/300
65/65 [==============================] - 14s 219ms/step - loss: 0.28
57 - accuracy: 0.9476 - val_loss: 8.9555 - val_accuracy: 0.2299
Epoch 13/300
64/65 [============================>.] - ETA: 0s - loss: 0.2469 - ac
curacy: 0.9558
Epoch 00013: ReduceLROnPlateau reducing learning rate to 0.000500000
0237487257.
65/65 [==============================] - 14s 220ms/step - loss: 0.24
66 - accuracy: 0.9558 - val_loss: 9.4296 - val_accuracy: 0.2132
Epoch 14/300
65/65 [==============================] - 14s 219ms/step - loss: 0.19
77 - accuracy: 0.9688 - val_loss: 9.7633 - val_accuracy: 0.2020
Epoch 15/300
65/65 [==============================] - 14s 223ms/step - loss: 0.18
25 - accuracy: 0.9750 - val_loss: 9.7320 - val_accuracy: 0.2249
Epoch 16/300
65/65 [==============================] - 15s 232ms/step - loss: 0.16
30 - accuracy: 0.9772 - val_loss: 9.8143 - val_accuracy: 0.2232
Epoch 17/300
65/65 [==============================] - 14s 221ms/step - loss: 0.14
93 - accuracy: 0.9834 - val_loss: 9.8631 - val_accuracy: 0.2227
Epoch 18/300
65/65 [==============================] - 14s 219ms/step - loss: 0.14
46 - accuracy: 0.9793 - val_loss: 9.8573 - val_accuracy: 0.2243
Epoch 19/300
```

```
65/65 [==============================] - 15s 226ms/step - loss: 0.12
76 - accuracy: 0.9853 - val_loss: 9.9643 - val_accuracy: 0.2227
Epoch 20/300
64/65 [============================>.] - ETA: 0s - loss: 0.1310 - ac
curacy: 0.9854
Epoch 00020: ReduceLROnPlateau reducing learning rate to 0.000250000
0118743628.
65/65 [==============================] - 15s 234ms/step - loss: 0.13
08 - accuracy: 0.9853 - val_loss: 10.2205 - val_accuracy: 0.2238
Epoch 21/300
65/65 [==============================] - 15s 225ms/step - loss: 0.11
65 - accuracy: 0.9894 - val_loss: 10.4129 - val_accuracy: 0.2171
Epoch 22/300
65/65 [==============================] - 15s 227ms/step - loss: 0.10
87 - accuracy: 0.9899 - val_loss: 10.4946 - val_accuracy: 0.2243
Epoch 23/300
65/65 [==============================] - 15s 226ms/step - loss: 0.10
27 - accuracy: 0.9901 - val_loss: 10.3977 - val_accuracy: 0.2266
Epoch 24/300
65/65 [==============================] - 15s 232ms/step - loss: 0.10
33 - accuracy: 0.9897 - val_loss: 10.7430 - val_accuracy: 0.2243
Epoch 25/300
65/65 [==============================] - 14s 223ms/step - loss: 0.09
61 - accuracy: 0.9911 - val_loss: 10.9743 - val_accuracy: 0.2204
Epoch 26/300
65/65 [==============================] - 15s 233ms/step - loss: 0.09
34 - accuracy: 0.9937 - val_loss: 10.5795 - val_accuracy: 0.2238
Epoch 27/300
64/65 [============================>.] - ETA: 0s - loss: 0.0910 - ac
curacy: 0.9932
Epoch 00027: ReduceLROnPlateau reducing learning rate to 0.000125000
0059371814.
65/65 [==============================] - 14s 220ms/step - loss: 0.09
06 - accuracy: 0.9933 - val_loss: 10.7604 - val_accuracy: 0.2254
Epoch 28/300
65/65 [==============================] - 15s 227ms/step - loss: 0.08
53 - accuracy: 0.9921 - val_loss: 10.9573 - val_accuracy: 0.2204
Epoch 29/300
65/65 [==============================] - 15s 229ms/step - loss: 0.08
54 - accuracy: 0.9923 - val_loss: 10.8397 - val_accuracy: 0.2254
Epoch 30/300
65/65 [==============================] - 15s 224ms/step - loss: 0.08
25 - accuracy: 0.9945 - val_loss: 10.9308 - val_accuracy: 0.2193
Epoch 31/300
65/65 [==============================] - 15s 235ms/step - loss: 0.08
25 - accuracy: 0.9918 - val_loss: 10.9023 - val_accuracy: 0.2215
Epoch 32/300
65/65 [==============================] - 15s 226ms/step - loss: 0.08
16 - accuracy: 0.9928 - val_loss: 11.0240 - val_accuracy: 0.2210
Epoch 33/300
65/65 [==============================] - 15s 236ms/step - loss: 0.07
99 - accuracy: 0.9935 - val_loss: 11.0105 - val_accuracy: 0.2232
Epoch 34/300
64/65 [============================>.] - ETA: 0s - loss: 0.0782 - ac
curacy: 0.9949
Epoch 00034: ReduceLROnPlateau reducing learning rate to 6.250000296
85907e-05.
65/65 [==============================] - 14s 222ms/step - loss: 0.07
83 - accuracy: 0.9947 - val_loss: 11.2312 - val_accuracy: 0.2193
Epoch 35/300
65/65 [==============================] - 15s 236ms/step - loss: 0.07
```

```
46 - accuracy: 0.9945 - val_loss: 11.1148 - val_accuracy: 0.2227
Epoch 36/300
65/65 [==============================] - 15s 224ms/step - loss: 0.07
27 - accuracy: 0.9964 - val_loss: 10.9853 - val_accuracy: 0.2243
Epoch 37/300
65/65 [==============================] - 15s 235ms/step - loss: 0.07
01 - accuracy: 0.9957 - val_loss: 11.1028 - val_accuracy: 0.2232
Epoch 38/300
65/65 [==============================] - 14s 220ms/step - loss: 0.07
50 - accuracy: 0.9947 - val_loss: 11.0248 - val_accuracy: 0.2266
Epoch 39/300
65/65 [==============================] - 15s 233ms/step - loss: 0.07
25 - accuracy: 0.9947 - val_loss: 11.1039 - val_accuracy: 0.2238
Epoch 40/300
65/65 [==============================] - 14s 222ms/step - loss: 0.07
32 - accuracy: 0.9966 - val_loss: 11.0651 - val_accuracy: 0.2227
Epoch 41/300
64/65 [============================>.] - ETA: 0s - loss: 0.0693 - ac
curacy: 0.9951
Epoch 00041: ReduceLROnPlateau reducing learning rate to 3.125000148
429535e-05.
65/65 [==============================] - 15s 229ms/step - loss: 0.06
94 - accuracy: 0.9952 - val_loss: 11.1655 - val_accuracy: 0.2232
Epoch 42/300
65/65 [==============================] - 15s 235ms/step - loss: 0.07
03 - accuracy: 0.9962 - val_loss: 11.1729 - val_accuracy: 0.2254
Epoch 43/300
64/65 [============================>.] - ETA: 0s - loss: 0.0701 - ac
curacy: 0.9954WARNING:tensorflow:Early stopping conditioned on metri
c `val_accuracy` which is not available. Available metrics are: los
s,accuracy
WARNING:tensorflow:Reduce LR on plateau conditioned on metric `val_a
ccuracy` which is not available. Available metrics are: loss,accurac
y,lr
65/65 [==============================] - 14s 212ms/step - loss: 0.07
00 - accuracy: 0.9954
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-80-ad09e789db27> in <module>
     10     steps_per_epoch=steps_per_epoch,
     11     validation_steps=validation_steps,
---> 12     callbacks=[tensorboard_callback, early_stopping, reduce_lr])

/usr/local/lib/python3.6/dist-packages/tensorflow_core/python/keras/engine/training.py in fit(self, x, y, batch_size, epochs, verbose, callbacks, validation_split, validation_data, shuffle, class_weight, sample_weight, initial_epoch, steps_per_epoch, validation_steps, validation_freq, max_queue_size, workers, use_multiprocessing, **kwargs)
    817         max_queue_size=max_queue_size,
    818         workers=workers,
--> 819         use_multiprocessing=use_multiprocessing)
    820 
    821   def evaluate(self,

/usr/local/lib/python3.6/dist-packages/tensorflow_core/python/keras/engine/training_v2.py in fit(self, model, x, y, batch_size, epochs, verbose, callbacks, validation_split, validation_data, shuffle, class_weight, sample_weight, initial_epoch, steps_per_epoch, validation_steps, validation_freq, max_queue_size, workers, use_multiprocessing, **kwargs)
    393                       mode=ModeKeys.TEST,
    394                       training_context=eval_context,
--> 395                       total_epochs=1)
    396                   cbks.make_logs(model, epoch_logs, eval_result, ModeKeys.TEST,
    397                                  prefix='val_')

/usr/local/lib/python3.6/dist-packages/tensorflow_core/python/keras/engine/training_v2.py in run_one_epoch(model, iterator, execution_function, dataset_size, batch_size, strategy, steps_per_epoch, num_samples, mode, training_context, total_epochs)
    126         step=step, mode=mode, size=current_batch_size) as batch_logs:
    127       try:
--> 128         batch_outs = execution_function(iterator)
    129       except (StopIteration, errors.OutOfRangeError):
    130         # TODO(kaftan): File bug about tf function and errors.OutOfRangeError?

/usr/local/lib/python3.6/dist-packages/tensorflow_core/python/keras/engine/training_v2_utils.py in execution_function(input_fn)
     96     # `numpy` translates Tensors to values in Eager mode.
     97     return nest.map_structure(_non_none_constant_value,
---> 98                               distributed_function(input_fn))
     99 
    100   return execution_function

/usr/local/lib/python3.6/dist-packages/tensorflow_core/python/eager/def_function.py in __call__(self, *args, **kwds)
    566         xla_context.Exit()
    567     else:
--> 568       result = self._call(*args, **kwds)
```

```
    569
    570     if tracing_count == self._get_tracing_count():

/usr/local/lib/python3.6/dist-packages/tensorflow_core/python/eager/
def_function.py in _call(self, *args, **kwds)
    604       # In this case we have not created variables on the fi
rst call. So we can
    605       # run the first trace but we should fail if variables
 are created.
--> 606       results = self._stateful_fn(*args, **kwds)
    607       if self._created_variables:
    608         raise ValueError("Creating variables on a non-first
 call to a function"

/usr/local/lib/python3.6/dist-packages/tensorflow_core/python/eager/
function.py in __call__(self, *args, **kwargs)
   2361     with self._lock:
   2362       graph_function, args, kwargs = self._maybe_define_func
tion(args, kwargs)
-> 2363     return graph_function._filtered_call(args, kwargs)  # py
lint: disable=protected-access
   2364
   2365   @property

/usr/local/lib/python3.6/dist-packages/tensorflow_core/python/eager/
function.py in _filtered_call(self, args, kwargs)
   1609          if isinstance(t, (ops.Tensor,
   1610                             resource_variable_ops.BaseResourc
eVariable))),
-> 1611         self.captured_inputs)
   1612
   1613   def _call_flat(self, args, captured_inputs, cancellation_m
anager=None):

/usr/local/lib/python3.6/dist-packages/tensorflow_core/python/eager/
function.py in _call_flat(self, args, captured_inputs, cancellation_
manager)
   1690       # No tape is watching; skip to running the function.
   1691       return self._build_call_outputs(self._inference_functi
on.call(
-> 1692           ctx, args, cancellation_manager=cancellation_manag
er))
   1693     forward_backward = self._select_forward_and_backward_fun
ctions(
   1694         args,

/usr/local/lib/python3.6/dist-packages/tensorflow_core/python/eager/
function.py in call(self, ctx, args, cancellation_manager)
    543               inputs=args,
    544               attrs=("executor_type", executor_type, "config
_proto", config),
--> 545               ctx=ctx)
    546         else:
    547           outputs = execute.execute_with_cancellation(

/usr/local/lib/python3.6/dist-packages/tensorflow_core/python/eager/
execute.py in quick_execute(op_name, num_outputs, inputs, attrs, ct
x, name)
     59     tensors = pywrap_tensorflow.TFE_Py_Execute(ctx._handle,
 device_name,
     60                                                op_name, inpu
```

```
ts, attrs,
---> 61                                                 num_outputs)
     62   except core._NotOkStatusException as e:
     63     if name is not None:

KeyboardInterrupt:
```

In [61]:

```
resnet.save('/data/rvgorb/hw10/resnet-50-fine-tuned')
```

```
INFO:tensorflow:Assets written to: /data/rvgorb/hw10/resnet-50-fine-
tuned/assets
```

In [63]:

```
test_accuracy = resnet.evaluate(ds_test)[1]
```

```
40/40 [==============================] - 6s 152ms/step - loss: 2.106
7 - accuracy: 0.4662
```

In [64]:

```
# Results
print("Итоговый результат:")
print("  test accuracy:\t\t{:.2f} %".format(
    test_accuracy * 100))

if test_accuracy * 100 > 40:
    print("Achievement unlocked: Transformer!")
elif test_accuracy * 100 > 35:
    print("Achievement unlocked: LSTM!")
elif test_accuracy * 100 > 30:
    print("Achievement unlocked: RNN!")
elif test_accuracy * 100 > 25:
    print("Achievement unlocked: perceptron!")
else:
    print("We need more \"layers\"! Follow instructons below")
```

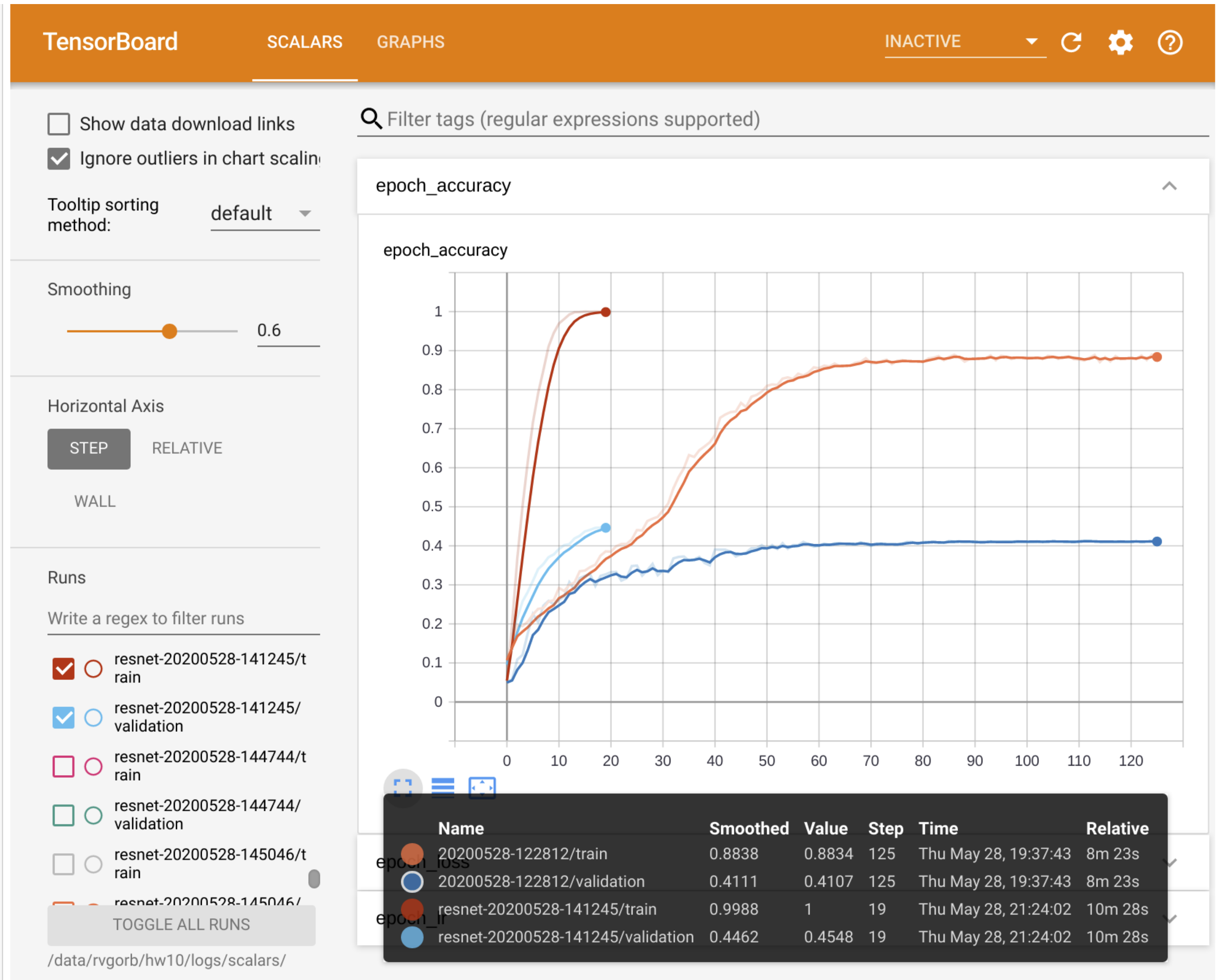
```
Итоговый результат:
  test accuracy:                46.62 %
Achievement unlocked: Transformer!
```

Comparing with my model:

In [81]:

```
tensorboard --logdir /data/rvgorb/hw10/logs/scalars/ --port 7775
```

```
Reusing TensorBoard on port 7775 (pid 5820), started 3:50:50 ago. (U
se '!kill 5820' to kill it.)
```



Conclusions:

As you can see, using *transfer learning* with minimum effort we achieved better results, so when you solving new task, a good way to do it is to find similar task with pretrained model and fine-tune it.