



# TensorFlow

Курс “Практическое применение по TensorFlow”

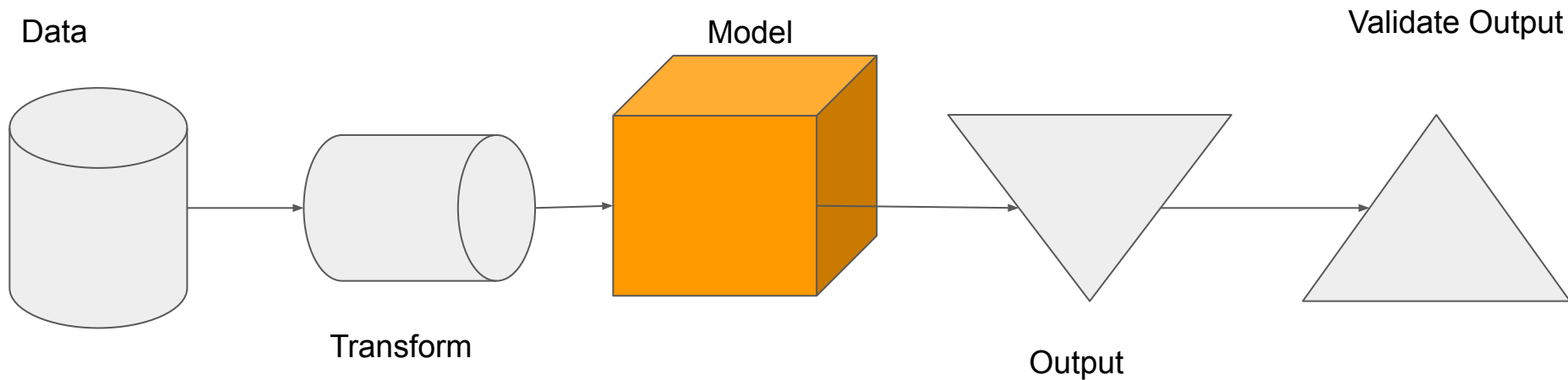
Шигапова Фирюза Зинатуллаевна

1-й семестр, 2019 г.



<https://github.com/Firyuza/TensorFlowPractice>

# Common pipeline of problem



# TensorFlow 1.x program

## Training ResNet50 for image classification

1. Load dataset via `keras.datasets`
2. Data Input Pipeline through TF `FIFOQueue`
3. Train ResNet50 via `keras.applications`
4. Log into Tensorboard
5. **Validate results (Homework)**


# TensorFlow 1.x program. Create graph

## 1. Declare placeholders

```
def __create_placeholders(self):
    self.image_placeholder = tf.placeholder(dtype=tf.float32, shape=(None,
                                                                    cfg.train.image_size,
                                                                    cfg.train.image_size,
                                                                    cfg.train.image_channels), name='image')
    self.image_label_placeholder = tf.placeholder(dtype=tf.int64, shape=(None), name='image_label')
    self.batch_size_placeholder = tf.placeholder(tf.int32, name='batch_size')
    self.augmentation_probability = tf.placeholder(dtype=tf.float32)
    self.phase_train_placeholder = tf.placeholder(dtype=tf.bool)

    self.global_step = tf.Variable(0, trainable=False)

    return
```



For counting passed iterations

# TensorFlow 1.x program. Create graph

## 2. Define process for loading data into FIFOQueue

augmentation  
probability for  
image  
transformations

```
def __prepare_batch(self, images, labels, augmentation_probability):
    elems = tf.convert_to_tensor([False, True], dtype=tf.bool)
    probs = tf.convert_to_tensor([1.0 - augmentation_probability, augmentation_probability])
    rescaled_probs = tf.expand_dims(tf.log(probs), 0)
    indices = tf.multinomial(rescaled_probs, tf.shape(images)[0])
    augment_probs = tf.gather(elems, tf.squeeze(indices, [0]))

    input_queue = tf.FIFOQueue(capacity=100000,
                              dtypes=[tf.float32, tf.int64, tf.bool],
                              shapes=[(cfg.train.image_size,
                                       cfg.train.image_size,
                                       cfg.train.image_channels), (), ()])
    enqueue_op = input_queue.enqueue_many([images, labels, augment_probs])

    nrof_preprocess_threads = cfg.train.nrof_threads
    images_and_labels = []
    for _ in range(nrof_preprocess_threads):
        image, label, augment_prob = input_queue.dequeue()
        image = self.__process_image(image)
        images_and_labels.append([image, label])

    image_batch, label_batch = tf.train.batch_join(images_and_labels,
                                                    batch_size=self.batch_size_placeholder,
                                                    enqueue_many=False)

    return enqueue_op, image_batch, label_batch
```

# TensorFlow 1.x program. Create graph

## 3. Create model

```
self.backbone_model = tf.keras.applications.resnet50.ResNet50(input_tensor=self.images_batch,  
                                                                weights='imagenet',  
                                                                include_top=False)  
backbone_output = self.backbone_model.output  
backbone_output = tf.keras.layers.GlobalAveragePooling2D()(backbone_output)  
self.logits = tf.keras.layers.Dense(self.nrof_classes)(backbone_output)
```

# TensorFlow 1.x program. Create graph

#### 4. Define Loss function that solve your optimization problem

[illegible]



# TensorFlow 1.x program. Create graph

5. Define optimizer -- does backward propagation and define the rule for updating trainable variables

```
opt = tf.train.AdamOptimizer(cfg.train.learning_rate, beta1=0.9, beta2=0.999)  
  
self.train_op = opt.minimize(self.total_loss, self.global_step, var_list=self.trainable_variables)
```

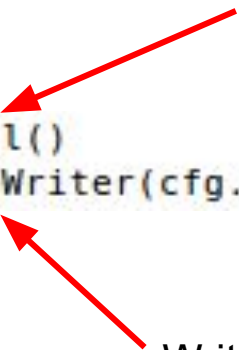
6. Run **self.train\_op** each training step through **session.run**

# TensorFlow 1.x program. Tensorboard

```
tf.summary.scalar('train_accuracy', self.tf_accuracy)
tf.summary.scalar('train_top5_accuracy', self.tf_top5_accuracy)
```

```
tf.summary.scalar('cross_entropy', self.loss)
tf.summary.scalar('reg_loss', self.reg_loss)
tf.summary.scalar('total_loss', self.total_loss)
tf.summary.scalar('learning_rate', self.learning_rate)
```

```
self.summary_op = tf.summary.merge_all()
self.summary_writer = tf.summary.FileWriter(cfg.train.logs_base_dir, self.sess.graph)
```



Op that generates all summaries data together

Write summaries data to the disk

# TensorFlow 1.x program. Save model

1. Create within graph context Saver:

```
self.saver = tf.train.Saver(tf.global_variables(), max_to_keep=None)
```

2. Save at needed time:

```
checkpoint_path = os.path.join(model_dir, 'model.ckpt')  
self.saver.save(self.session, checkpoint_path, global_step=step)
```

3. Restore model:

```
self.saver.restore(self.session, cfg.train.restore_model_path)
```

# TensorFlow 1.x program. Training step

1. Think over Data Input Pipeline (how to put and take data each step/epoch)
2. Create loop where you can track each step if it's needed
3. Log results

# TensorFlow 1.x program. Dataset

```
(self.x_train, self.y_train), (self.x_test, self.y_test) =  
    tf.keras.datasets.cifar10.load_data()
```

# TensorFlow 1.x program. Check Tensorboard

In terminal write down:

**tensorboard --logdir=<*path to your log dir*>**

SCALARS

GRAPHS

d links

art scaling

default

0

WALL

RUNS

jects/  
nf/loss

0 400 800 1.2k 1.6k 2k



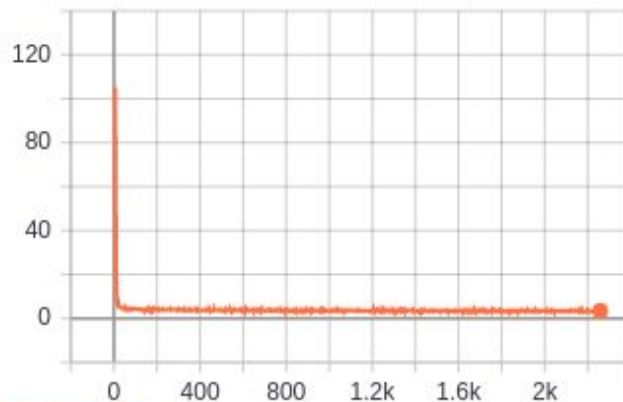
cross\_entropy

learning\_rate

reg\_loss

total\_loss

total\_loss



# TensorFlow 1.x program. Graph View

