GOETHE UNIVERSITY FRANKFURT AM MAIN

Deep Learning on Kubeflow

Iurii Mozzhurin & Rebekka Pech

Hands-on Lab - Big Data Technologies SS 2019
July 28th, 2019

1. Introduction to Kubeflow

Kubeflow is an open source platform which is based on Kubernetes and was developed by Google. It's dedicated for making deployments of Machine Learning workflows on Kubernetes simple, portable (it works on any Kubernetes cluster) and scalable (upon need). Kubeflow makes it possible to bring ML in any existing cluster.

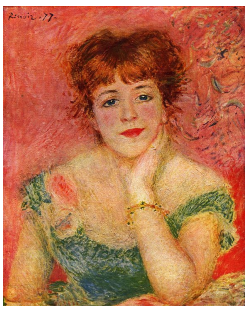
When talking about Kubeflow you also have to talk about Kubernetes. Kubernetes is an open-source container-orchestration system. With Kubernetes you only need to define what resources the app needs, how many instances should be made, and Kubernetes takes care of deploying, managing and balancing containers on the hardware (nodes).

The main idea of Kubeflow is that it's a ready-to-use set of Machine Learning tools. It's not necessary to install and connect the tools separately, but it can be done automatically. It also provides GUI, which makes the work with them easy for non-programmers.
Kubeflow includes components for model developing, training and serving and also for hyperparameter tuning and pipelines.
Kubeflow should be used for distributed ML workflow, for ML in production or for ML in cloud.

2. Use Case Setup

The intention of our use case was to train a CNN model to distinguish between works of ten painters. In the end we wanted to have a (good) prediction when we give an image into our model whose painting it is.

For this model we used the Dataset "Painter by Numbers" from kaggle.com which has an original size of 83GB but that includes a lot of images from much more than 10 artists. Most of the images were from WikiArt.org.
In the original dataset we had Image dimensions from 477 to 7444 Pixels.



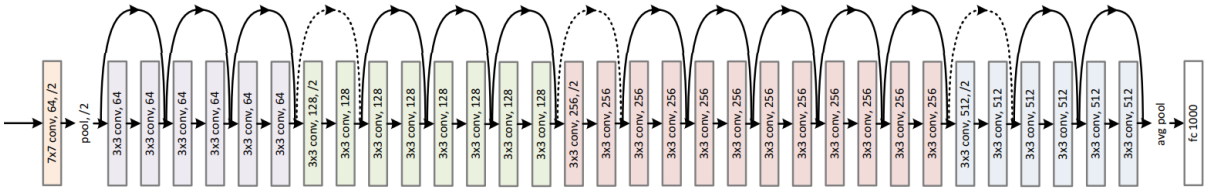
Example: Painting of *P.Renoir*

We selected a small part of this data and also resized it to 448 Pixels on the smallest side. Therefor in the end we worked with 340 MB. By that we had 388 images per artist from 10 different painters. From these images we used 20 images per artist for one painter for testing - that's 5 percent of all data - and the rest was used for the training of the model.

ResNet-50.

For the training of our model we used *ResNet-50* which is a pretrained neural network. This network is trained on more than a million images from the *ImageNet Database*. The mage input size of the network is 224-by-224 pixels.

The network is 50 layers deep and can classify images into 1000 object categories. For the implementation of our use case we just changed the last layer with the training from our model for the ten classes.



We also used image augmentation for the model by using shift, rotation and flip for our images. These random transformations should make the dataset larger and make sure that you have the same images in different angles.

Milestones.

When we started our use case we made a plan what milestones we want to achieve. These are the objectives we had at the beginning:

1. Preprocess the data
2. Create a model for transfer learning in Python
3. Create a Docker image for this model
4. Traine the model on GKE using TensorFlowJobs on CPUs
5. Save the trained model
6. Traine the model on GKE using TensorFlowJobs on GPUs, because it's much faster (about 15 times faster)
7. Distributed training on multiple pods
8. Serving the model with TensorFlowServing

Technologies.

To create the model we used Tensorflow and Keras.
For our containers we used Docker and Kubernetes for organisation.
To manage the pods we first wanted to use kubectl and ksonnet. Meanwile they changed in Kubeflow ksonnet to kustomize but in the end they also deleted kustomize, so we just used kubectl alone.
To manage the Kubeflow Deployment we used the GCP console, gcould and gsutil.
The data we managed with PVC and saved it in a GCS bucket.
For monitoring we used Tensorboard.
In the end we managed the model with TensorFlow Serving in Jupyter Notebook.

3. Use Case Implementation

Create the model.

We created the model for transfer learning in Python.
Therefore we used our ResNet-50 network.

```
# Create the model for transfer learning using pretrained
    ResNet50
      print('>>>>>>>>>>>>>>>>>>>>>>>>')
      print('Creating the model...')
      model = Sequential()
```

```
model.add(ResNet50(
    include_top=False,
    weights=weights,
    pooling='avg'
))

model.add(Dense(
    args.numclasses,
    activation='softmax'
))

# Do not train the first layer in transfer learning
if args.transferlearning:
     model.layers[0].trainable = False

model.compile(
    optimizer=args.optimizer,
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

Create a Docker.

```
#This container contains your model and any helper scripts
   specific to your model.
FROM python:3.7
FROM tensorflow/tensorflow:1.13.1-py3

COPY requirements_io.txt /opt/
RUN pip install -r /opt/requirements_io.txt

ADD model.py /opt/model.py
RUN chmod +x /opt/model.py

ENTRYPOINT ["python"]
CMD ["/opt/model.py"]
```

Training.

For the training we first used image augmentation and then trained the model:

```
# Image generators
    image_size = 224
```

```
print('>>>>>>>>>>>>>>>>>>>>>>>>')
print('Creating the image generators...')
data_generator_no_aug = ImageDataGenerator(
    preprocessing_function=preprocess_input)

data_generator_with_aug = ImageDataGenerator(
    preprocessing_function=preprocess_input,
                                        horizontal_flip=bool(
                                            args.hflip),
                                        vertical_flip=bool(args.
                                            vflip),
                                        rotation_range=args.
                                            rotation,
                                        width_shift_range = args
                                            .wshift,
                                        height_shift_range =
                                            args.hshift)

train_generator_with_aug = data_generator_with_aug.
    flow_from_directory(
            working_train_dir,
            target_size=(image_size, image_size),
            batch_size=args.batchsize,
            class_mode='categorical')

validation_generator = data_generator_no_aug.
    flow_from_directory(
            working_test_dir,
            target_size=(image_size, image_size),
            batch_size=args.batchsize,
            class_mode='categorical')

# Train the model
print('>>>>>>>>>>>>>>>>>>>>>>>>')
print('Starting the training...')
history_aug = model.fit_generator(
            train_generator_with_aug,
            epochs=args.epochs,
            validation_data=validation_generator,
            shuffle=True,
            callbacks=[tensorboard])
```

Save the model.

With Keras we couldn't save the model directly to the bucket. So we saved it locally first and then send it to the bucket (copy2bucket).

```
print('>>>>>>>>>>>>>>>>>>>>>>>>')
print('Saiving the model localy...')
time_stamp = str(int(time.time()))
model_name = 'top' + str(args.numclasses) + '-' +
   time_stamp + '.h5'
model.save(model_name)


print('>>>>>>>>>>>>>>>>>>>>>>>>')
print('Trying to copy the model to a bucket...')
try:
    copy2bucket(model_name, args.exportdir)
except Exception as e:
    print(e)

print('>>>>>>>>>>>>>>>>>>>>>>>>')
print('Trying to copy the logs to a bucket...')
try:
    import pickle
    history_file = 'history-top' + str(args.numclasses) + '
      -' + time_stamp + '.pkl'
    with open(history_file, 'wb') as f:
        pickle.dump(history_aug.history, f)
    copy2bucket(history_file, args.exportdir)
    logs = os.listdir('/logs')
    for l in logs:
        copy2bucket('/logs'+l, args.exportdir)
except Exception as e:
    print(e)
```

The function that is used to copy the saved model to the bucket:

```
def copy2bucket(file, bucket):
    with file_io.FileIO(file, mode='rb') as i_f:
        with file_io.FileIO(os.path.join(bucket, file), mode='
          wb+') as o_f:
            o_f.write(i_f.read())
            print('Copied', file, 'to', bucket)
```

Results.

After the training you can check the predicitions. We ran it on Jupyter Notebook. Therefore we import the trained and saved model. Then we import the csv-file and select the top 10 painters.

Check the predictions

In [27]:
```
from tensorflow.python.keras import saving
trained_model = saving.load_model(model_path)
```

In [28]:
```
# Import CSV and select top 10 painters
df = pd.read_csv('../all_data_info.csv')
mask = df['in_train']
train_3_df = df[mask]
top10 = train_3_df.groupby('artist').count().sort_values(by='title', ascending=False).head(10)
top10.index
```

Out[28]:
```
Index(['Paul Cezanne', 'Giovanni Battista Piranesi', 'Martiros Saryan',
       'Ilya Repin', 'Camille Pissarro', 'Vincent van Gogh',
       'Theophile Steinlen', 'Pyotr Konchalovsky', 'Pierre-Auguste Renoir',
       'Boris Kustodiev'],
      dtype='object', name='artist')
```

Then we select a random image of the input data and resize it to 224-by-224 pixels.

In [29]:
```
# Chose a random image

data_dir = '/home/mo/Downloads/painters/'
train_dir = data_dir + 'top10_x448/'
image_size = 224

rand_top10 = list(top10.index)
random.shuffle(rand_top10)
true_artist = rand_top10[0]
pictures = list(train_3_df[(train_3_df['artist'] == true_artist)]['new_filename'])
img = random.choice(pictures)
image_name = train_dir + img
image = Image.open(image_name).resize((image_size, image_size))
print('Artist: ' + true_artist)
image
```

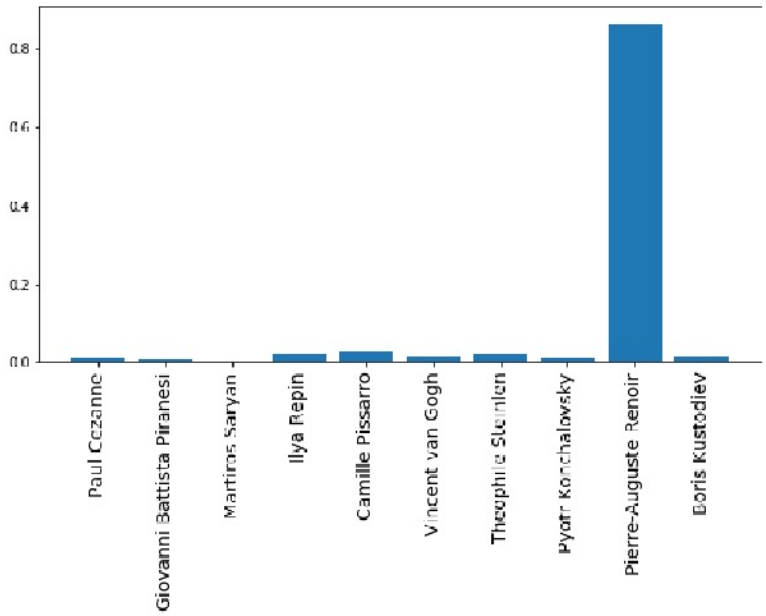
```
Artist: Pyotr Konchalovsky
```

Out[29]:

In the end we get a display for the predicition of the model where the probability for each of the ten painters is shown in percent.

```
In [30]: # Get predictions
         image = np.array(image)/255.0
         result = trained_model.predict(image[np.newaxis, ...])
         predicted_class = np.argmax(result[0], axis=-1)
```

```
In [34]: plt.figure(figsize=(10,5))
         plt.bar(range(10), result[0])
         plt.xticks(range(10), top10.index, rotation=90, fontsize=14)
         plt.show()
```



Our model doesnt't give a good prediction for the 10 painters and it always shows the same prediction independently of the input image. That's because we just used a small amount of data for training so the model cannot give a proper prediction.

Problems.

One of our Problems was that we used Keras, but Keras cannot save models/logs directly to the bucket. We had to save it locally and then send it to the bucket.
Also there is no obvious support for distributed learning on one computer we could use.

Another problem was that we first tried to use Kustomize and then Ksonnet but they didn't work anymore.

4. Conclusion

Keras simplifies the work with CNN models but it has several restrictions:
On the one hand Keras cannot work directly with GCS buckets. On the other hand Keras does not directly support distributed learning.

Tensorflow itself requires additional learning time if you want to use it.

In conclusion we can say that Kubeflow is still very raw. There are rapid changes and a low backward compatibility. Also you have to learn all underlying technologies like Kubernetes, Docker, Tensorflow.
Finally it's important to say that Kubeflow is a tool that is not for data science. It is for model training.

We only recommend it for machine learning and only for people that have experience or knowledge of the used tools in Kubeflow. Also we would recommend to use it more in the futur when Kubeflow is more sophisticated.

References

[1] https://www.kubeflow.org/docs/

[2] https://github.com/kubeflow/examples

[3] https://www.kaggle.com/c/painter-by-numbers/

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. *Deep Residual Learning for Image Recognition*, arXiv:1512.03385

[5] https://keras.io/

[6] https://www.tensorflow.org/guide

[7] https://kubernetes.io/docs/