AUTOMATING AI LIFECYCLE: THE DDOS USE CASE

Eros Zaupa

PROJECT GOALS

CICDDoS2019
: Explore a new dataset for DDoS attacks

DNN
: Develop a detection system using the dataset

Kubeflow
: Design and develop a ML pipeline

WHY

AI solutions have a typical lifecycle

Preprocessing
: Convert data into a suitable format for the problem under study

Hyper-parameter tuning
: Search the best model configuration for the task

Testing
: Measure the model performances on unseen data

Problem - Resource and time demanding

WHAT

Find a way to automate and speed up this lifecycle

Idea

Distribute the workload among different units

- Identify any independent part of the execution flow
- Parallelize tasks when possible

HOW

Kubeflow

Deployment of ML workflows on Kubernetes

- Toolkit for K8s
- Simple, portable and scalable
- Development, testing, and production-level serving

CICDDOS2019 - DATASET

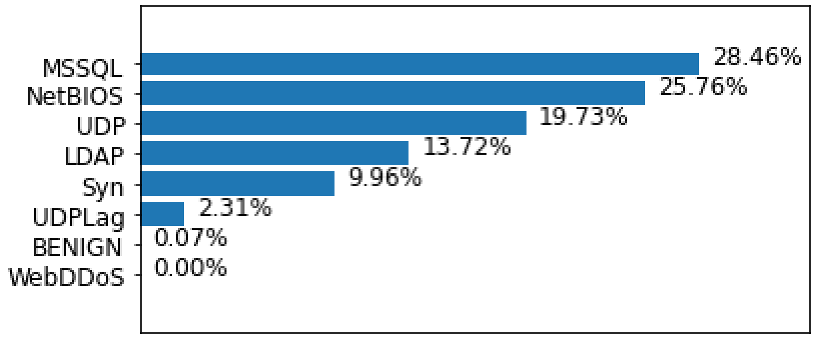
Raw data

With network traffic and event logs

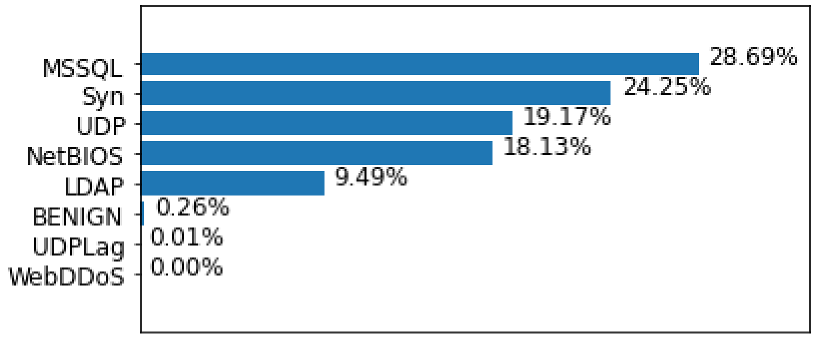
CSV files

More than 80 traffic features extracted from the raw data

DATASETS FOR DNN

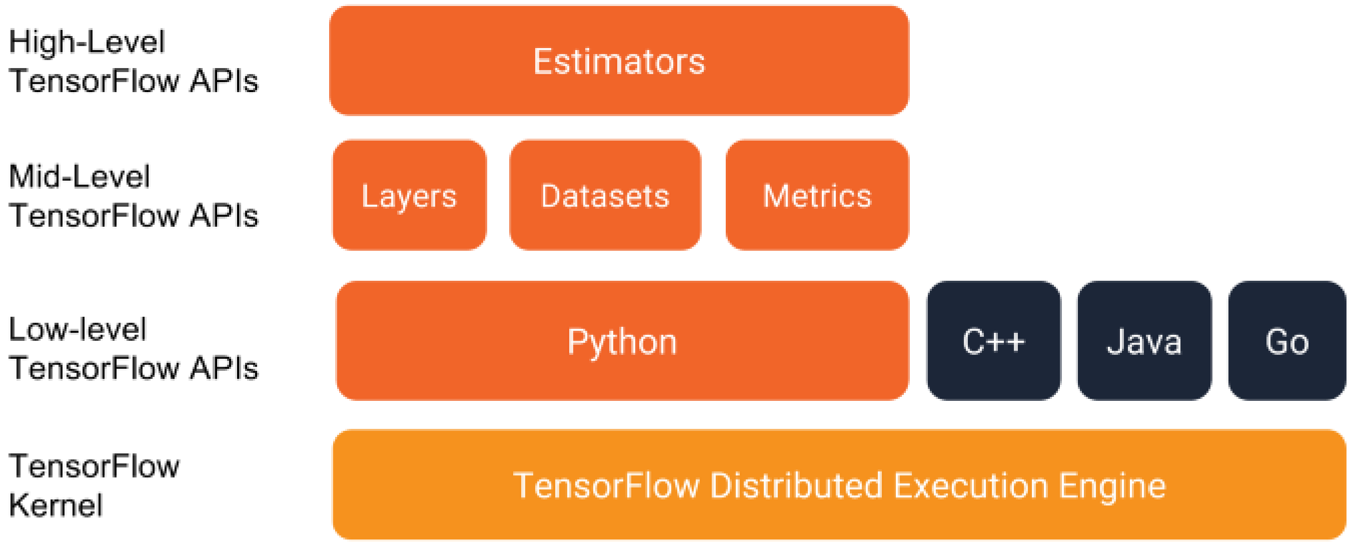


Training dataset



Testing dataset

TENSORFLOW ESTIMATORS


High-Level TensorFlow APIs
Estimators
Mid-Level TensorFlow APIs
Layers
Datasets
Metrics
Low-level TensorFlow APIs
Python
C++
Java
Go
TensorFlow Kernel
TensorFlow Distributed Execution Engine

Tensorflow API stack

DESIGN

- Network
    - Dense, feed-forward neural network
- Multiclassification
    - 8 classes
- Features
    - 20 most useful features
- Batch normalization
- Adam optimizer

HYPERPARAMETER TUNING

- Number of hidden units
  - [60, 30, 20]
  - [60, 40, 30, 20]
- Dropout rate
  - 0.1
  - 0.2
- Learning rate
  - 0.1
  - 0.3

PIPELINE DEVELOMENT

- Docker 18.09.7
- Kubernetes v1.15.3
- Kubeflow v1.0
    - Kubeflow Pipeline SDK v1.0.0

RESOURCES

Master node

4 VCPUs, 8GB RAM, 100GB of storage

2 x Slave nodes

4 VCPUs, 16GB RAM, 100GB of storage

OS

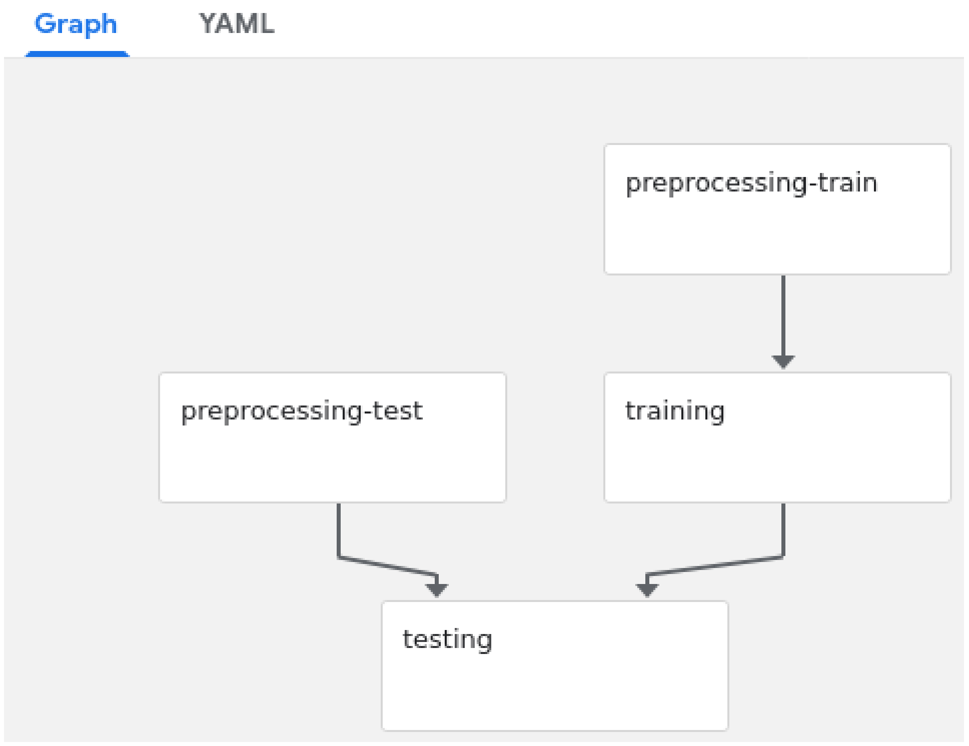
Ubuntu 16.04 LTS

PIPELINES

Description of an ML workflow, which

- Components, and how they combine in the form of a graph
- Inputs required for a run
- Inputs and outputs of each component

PIPELINES


Graph
YAML
preprocessing-train
preprocessing-test
training
testing

COMPONENTS

Base image
: All the shared dependencies

Preprocess-train
: Training dataset + Source code

Preprocess-test
: Testing dataset + Source code

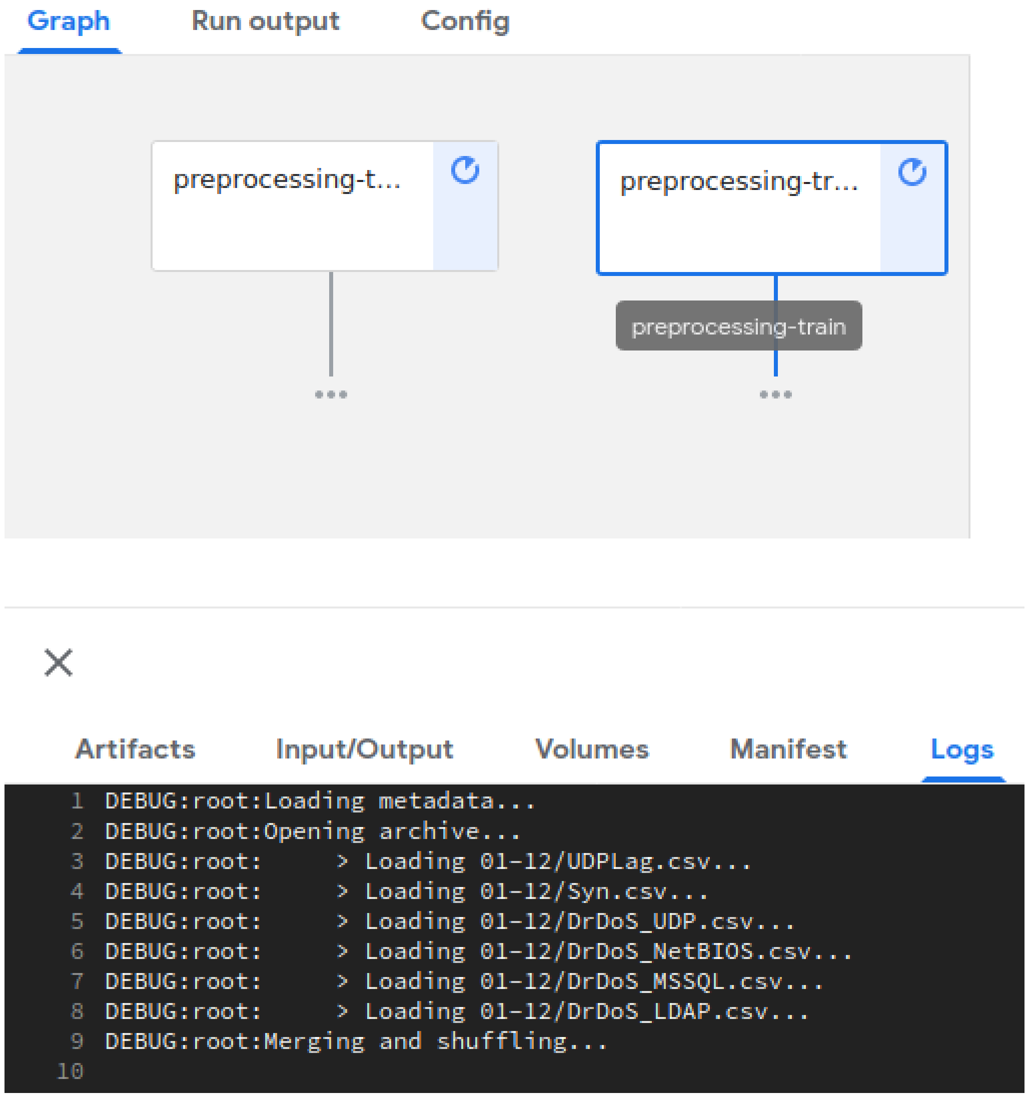
Train
: Source code

Test
: Source code

EXPERIMENTS

- Workspace to try different configurations of pipelines
- Organize runs into logical groups

EXPERIMENTS



```
DEBUG:root:Loading metadata...
DEBUG:root:Opening archive...
DEBUG:root:     > Loading 01-12/UDPLag.csv...
DEBUG:root:     > Loading 01-12/Syn.csv...
DEBUG:root:     > Loading 01-12/DrDoS_UDP.csv...
DEBUG:root:     > Loading 01-12/DrDoS_NetBIOS.csv...
DEBUG:root:     > Loading 01-12/DrDoS_MSSQL.csv...
DEBUG:root:     > Loading 01-12/DrDoS_LDAP.csv...
DEBUG:root:Merging and shuffling...
```

BEHAVIOUR

Load is distributed

Components are executed according to the available resources

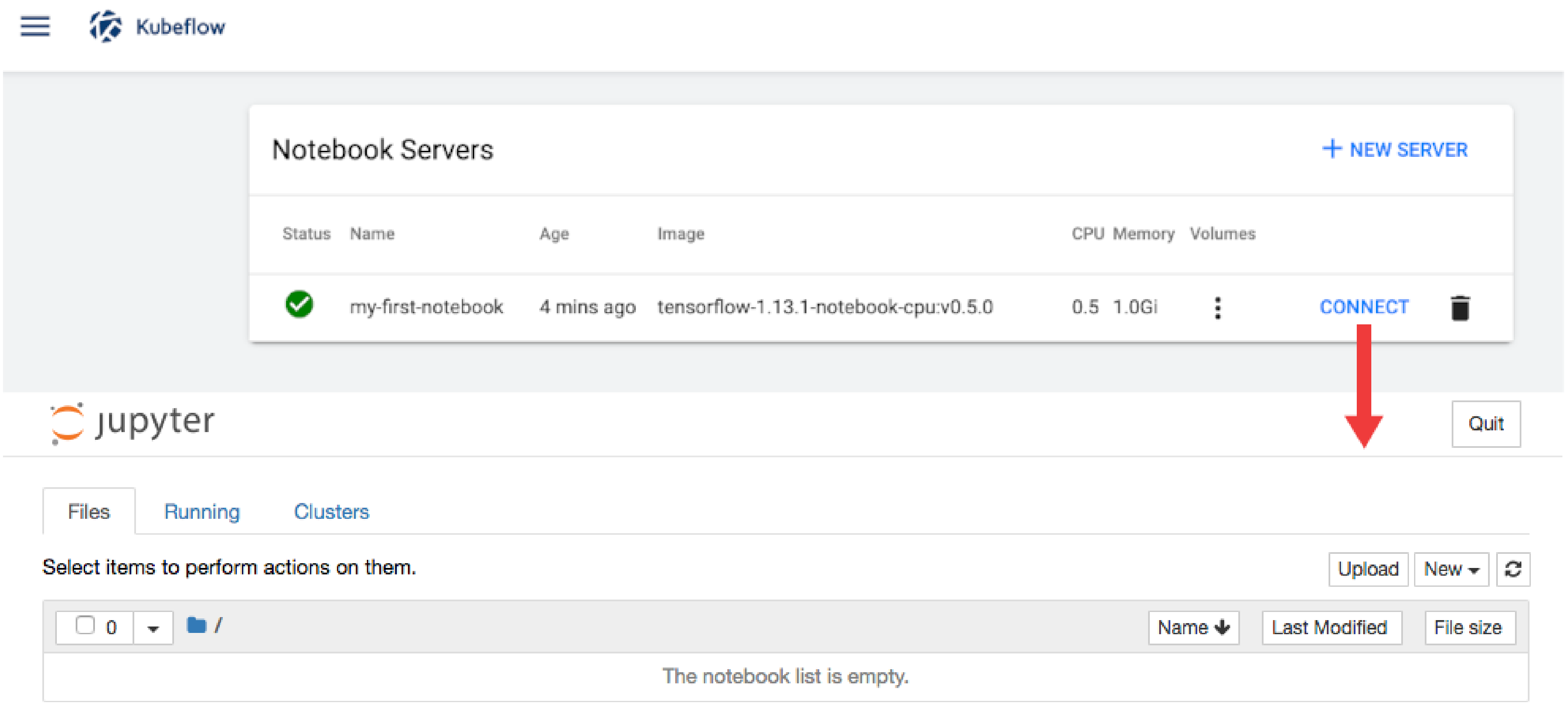
Failure

If any node fails, the experiment is resumed as soon as the node is again available

SOLUTION 1

- Jupyter notebook, implementing all the phases
- Run on a notebook server instance (2CPU, 10GB)

SOLUTION 1



Notebook Servers

+ NEW SERVER

Status	Name	Age	Image	CPU	Memory	Volumes		
✓	my-first-notebook	4 mins ago	tensorflow-1.13.1-notebook-cpu:v0.5.0	0.5	1.0Gi	⋮	CONNECT	

Select items to perform actions on them.

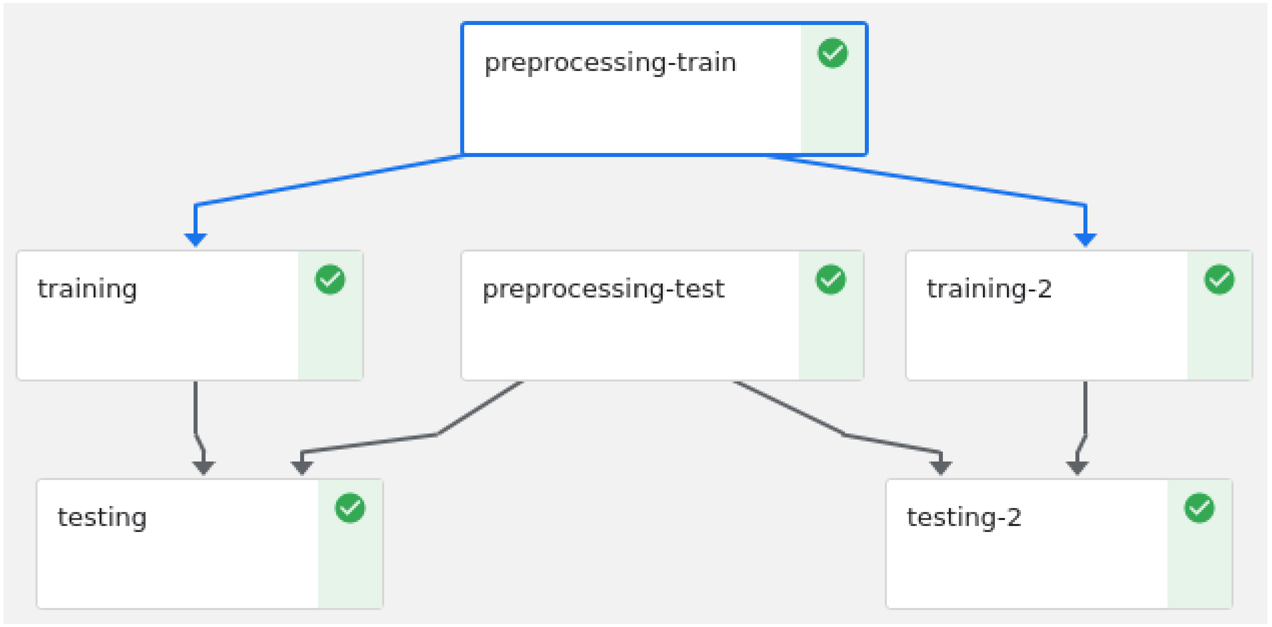
The notebook list is empty.

SOLUTION 2A

Concurrent, with two branches (with training and testing) executing the hyper-parameter tuning for dropout rate and learning rate

- Branch 1 on a [60, 30, 20] structure
- Branch 2 on a [60, 40, 30, 20] structure.

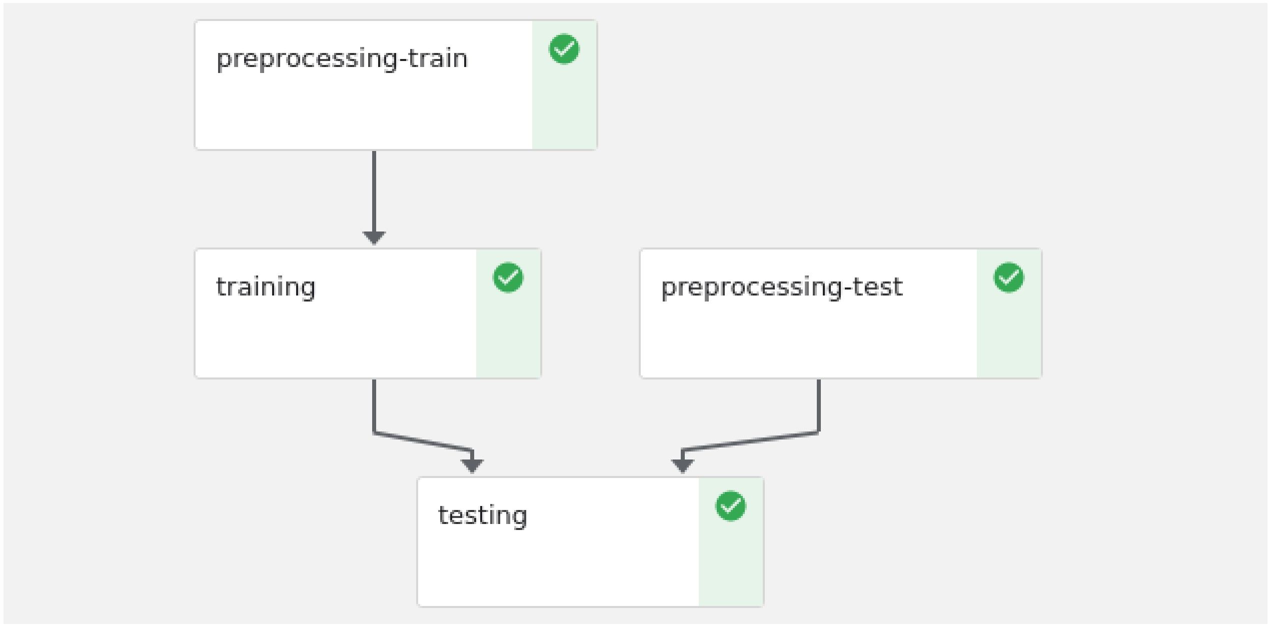
SOLUTION 2A


preprocessing-train
training
preprocessing-test
training-2
testing
testing-2

SOLUTION 2B

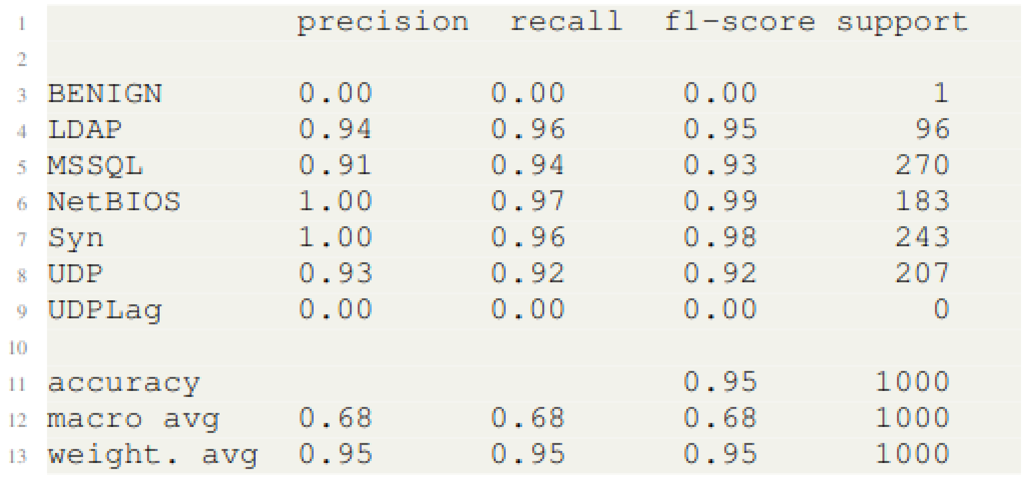
Non-concurrent, with just one branch that executes hyper-parameter tuning on number of hidden units, learning rate and dropout rate

SOLUTION 2B


preprocessing-train
training
preprocessing-test
testing

PERFORMANCE

$$Pr = \frac{TP}{TP + FP}, Rc = \frac{TP}{TP + FN}, F1 = \frac{2 * (Pr * Rc)}{Pr + Rc}$$



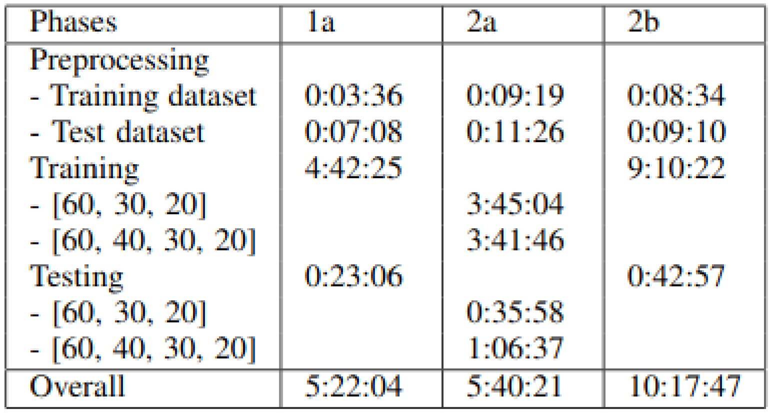
```
             precision  recall  f1-score  support

BENIGN         0.00      0.00     0.00          1
LDAP           0.94      0.96     0.95         96
MSSQL          0.91      0.94     0.93        270
NetBIOS        1.00      0.97     0.99        183
Syn            1.00      0.96     0.98        243
UDP            0.93      0.92     0.92        207
UDPLag         0.00      0.00     0.00          0

accuracy                          0.95       1000
macro avg      0.68      0.68     0.68       1000
weight. avg    0.95      0.95     0.95       1000
```

Listing 3. Example of a typical report executed over 1000 samples

TIMING



Phases	1a	2a	2b
Preprocessing			
- Training dataset	0:03:36	0:09:19	0:08:34
- Test dataset	0:07:08	0:11:26	0:09:10
Training	4:42:25		9:10:22
- [60, 30, 20]		3:45:04	
- [60, 40, 30, 20]		3:41:46	
Testing	0:23:06		0:42:57
- [60, 30, 20]		0:35:58	
- [60, 40, 30, 20]		1:06:37	
Overall	5:22:04	5:40:21	10:17:47

TABLE 2. TIMES COMPARISON BETWEEN DIFFERENT SOLUTIONS (EXPRESSED IN H:MM:SS)

COMMENTS

- Significant reductions in times with concurrency
- Small overhead on component initialization and management
- Pipeline implementations are overall slower than the notebook execution

  Warning

  Your CPU supports instructions that this TensorFlow binary was not compiled touse: SSE4.1 SSE4.2

CONCLUSIONS - DATASET

- Highly inbalanced
  - Deal with the inbalance (e.g. resampling)
- More potential to be discovered
  - Use of raw data (and custom exctraction of features)

CONCLUSIONS - KUBEFLOW

Portability, reusability, concurrency

- TensorFlow with full instruction set support
  - May significantly reduce training times
- Increase the level of concurrency
  - Scaling with the amount resources
- Kubeflow Katib for hyperparameter tuning
  - Beta/alpha stage, focus on optimization