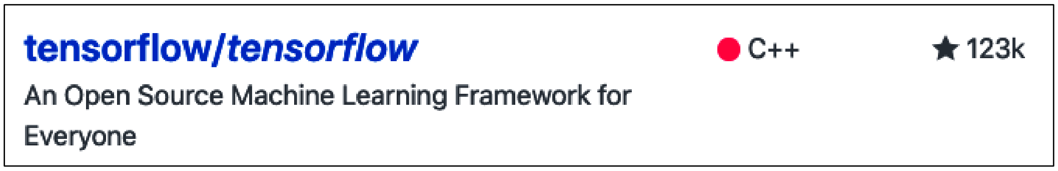
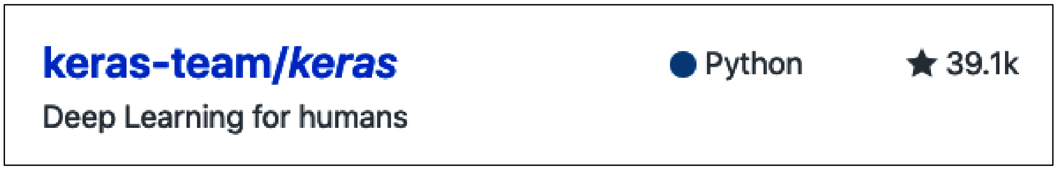
Chapter 1: Neural Network Foundations with TensorFlow 2.0



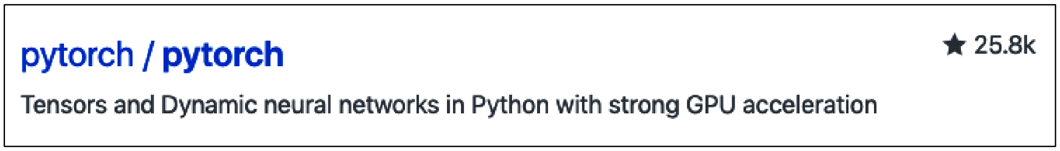
tensorflow/tensorflow ● C++ ★ 123k

An Open Source Machine Learning Framework for Everyone



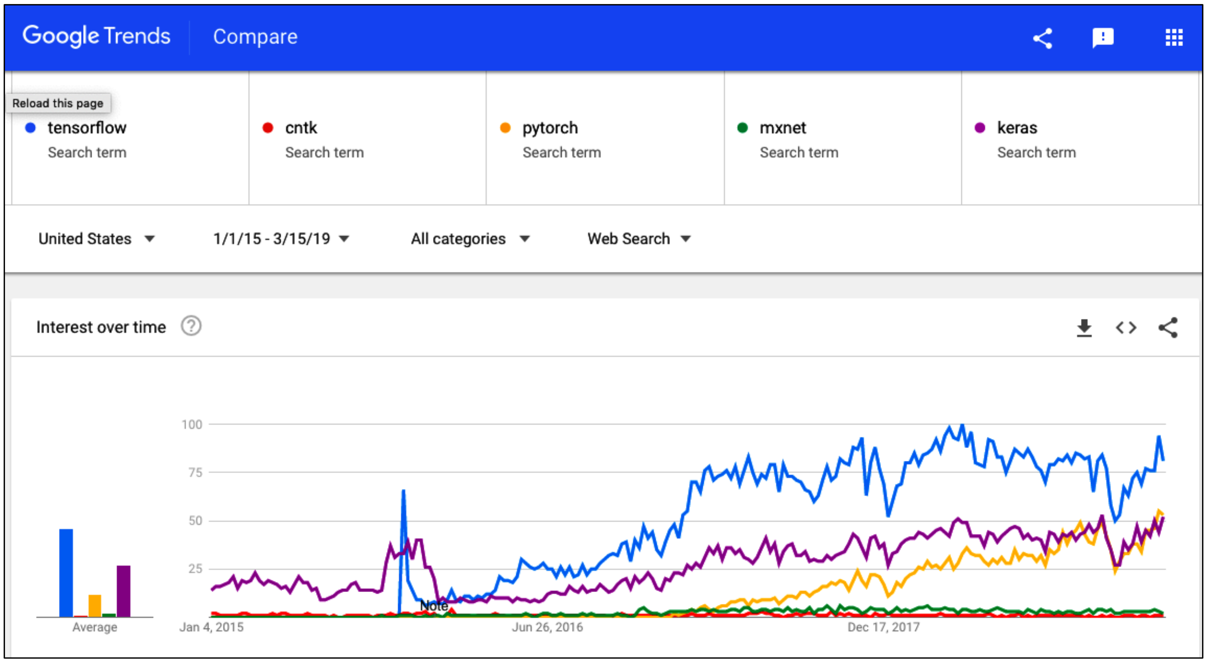
keras-team/keras ● Python ★ 39.1k

Deep Learning for humans



pytorch / pytorch ★ 25.8k

Tensors and Dynamic neural networks in Python with strong GPU acceleration



Google Trends Compare

Reload this page

● tensorflow
Search term

● cntk
Search term

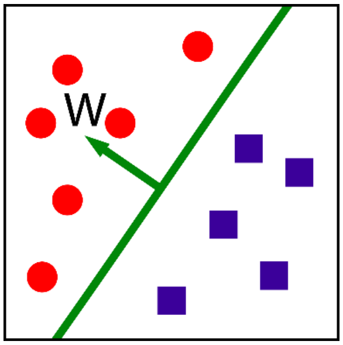
● pytorch
Search term

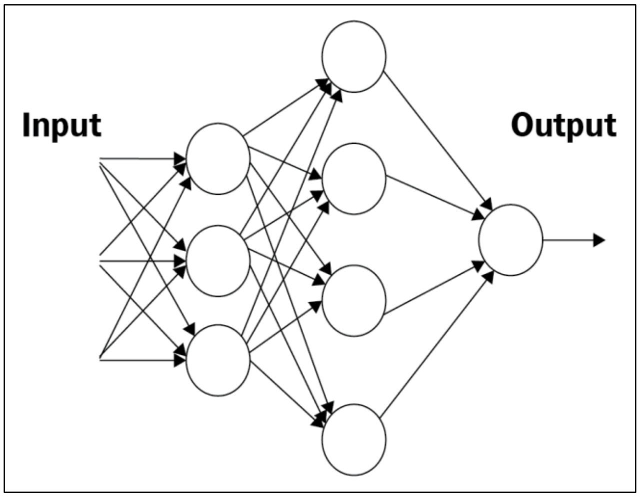
● mxnet
Search term

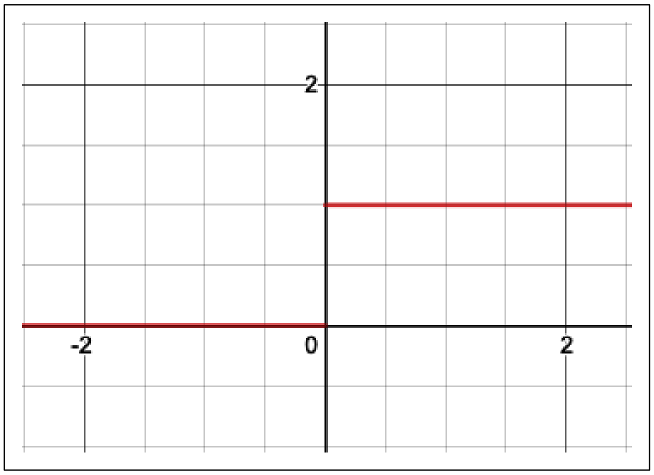
● keras
Search term

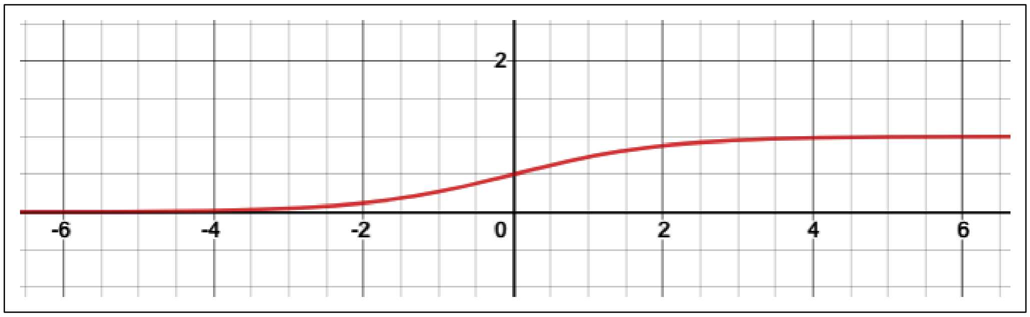
United States ▼ 1/1/15 - 3/15/19 ▼ All categories ▼ Web Search ▼

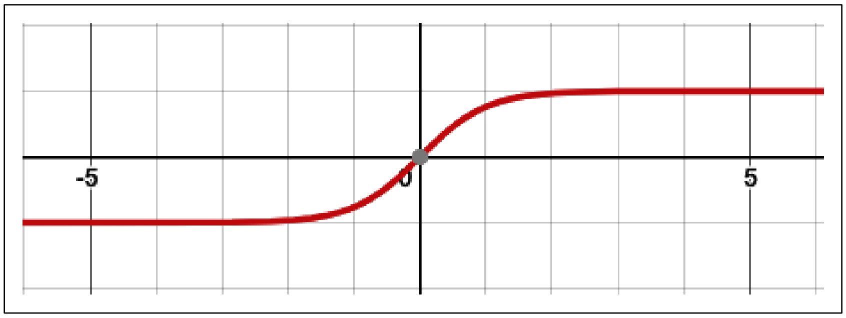
Interest over time

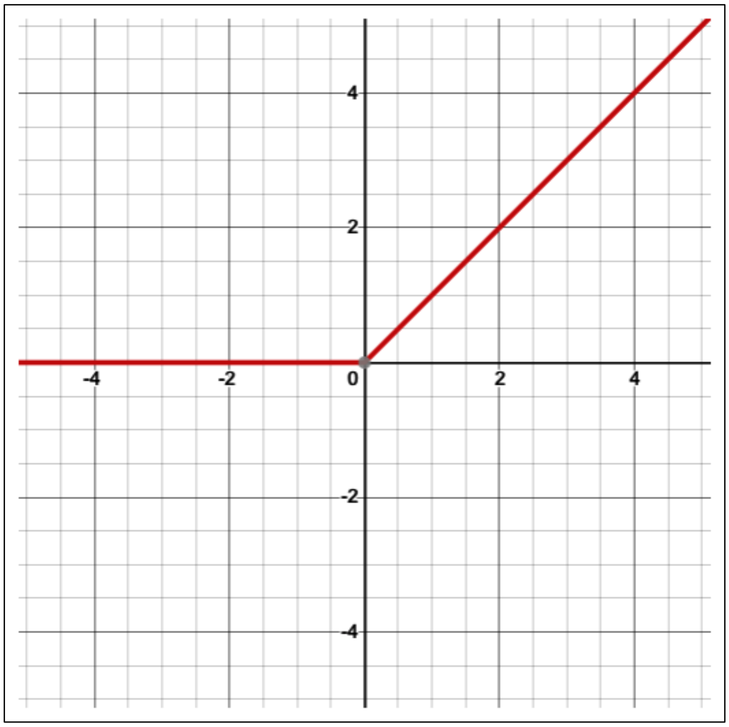

W

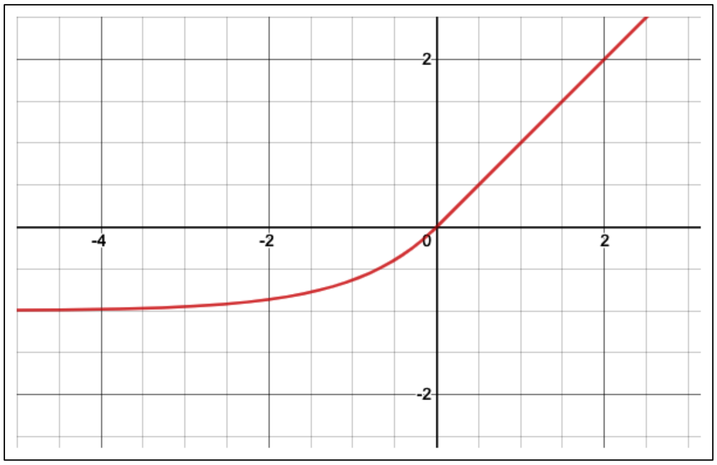

Input
Output

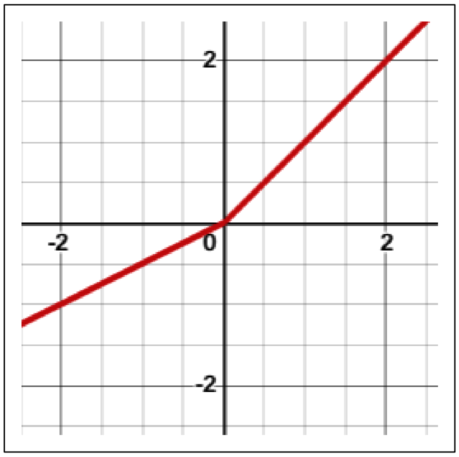

2
-2
0
2

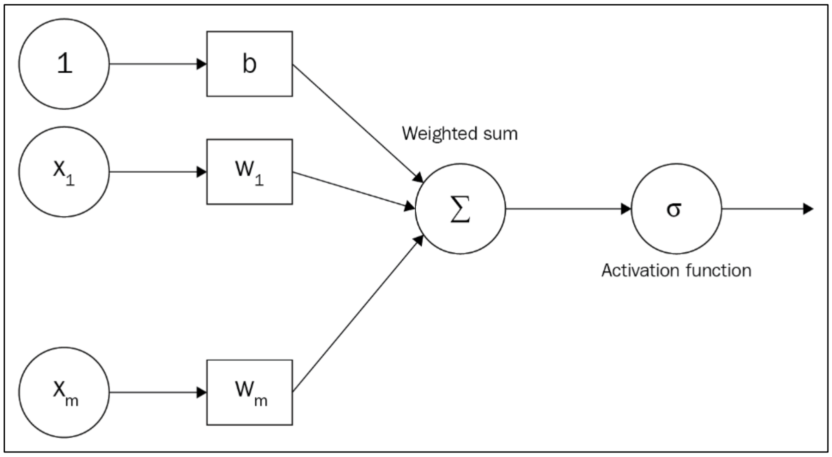

2
-6
-4
-2
0
2
4
6

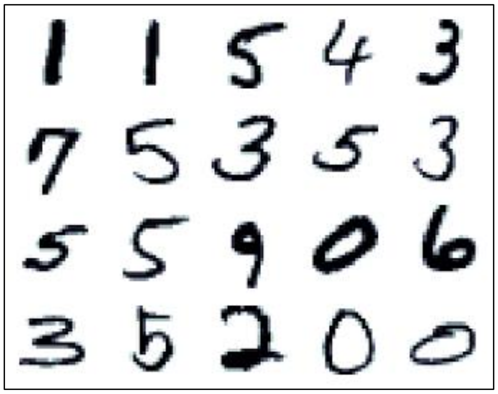

-5
0
5

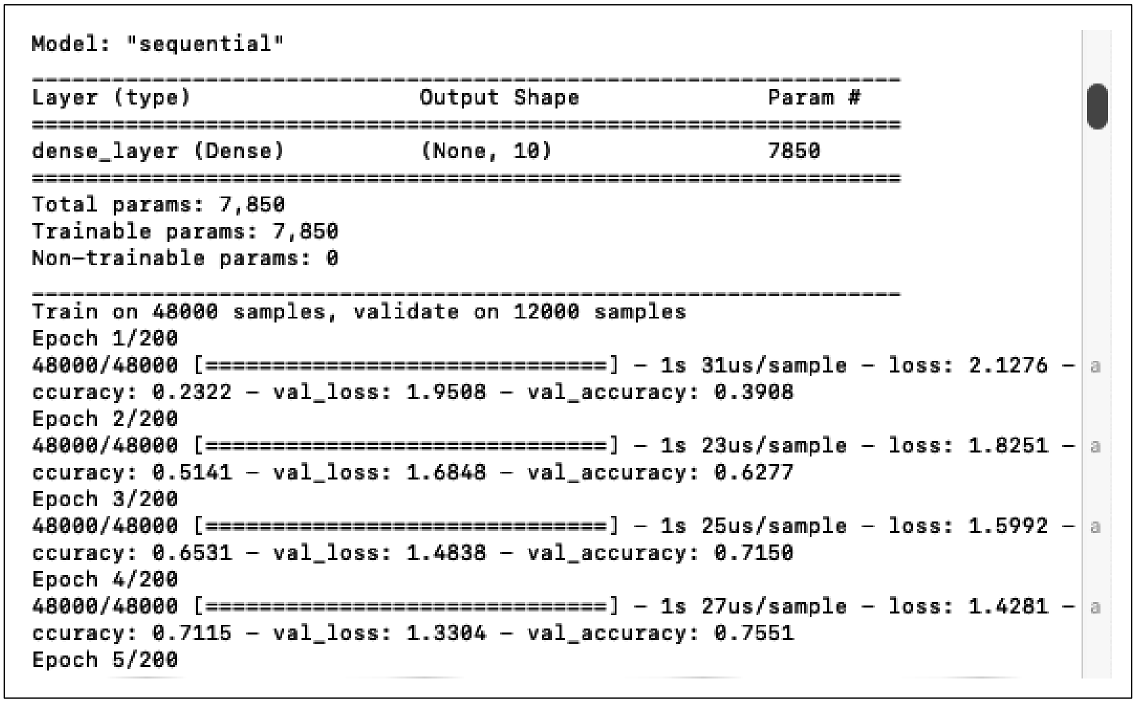

4
2
-4
-2
0
2
4
-2
-4


2
-4
-2
0
2
-2

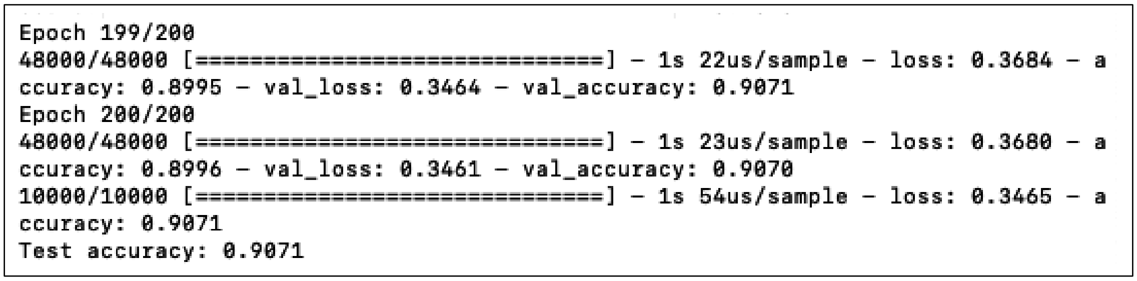

2
-2
0
2
-2


1
b
X_1
W_1
Weighted sum
Σ
σ
Activation function
X_m
W_m

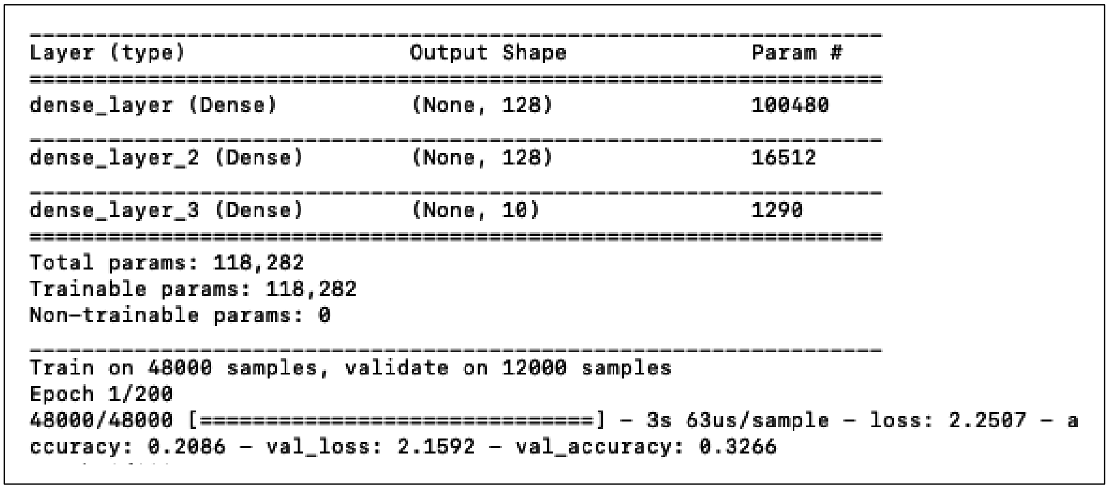




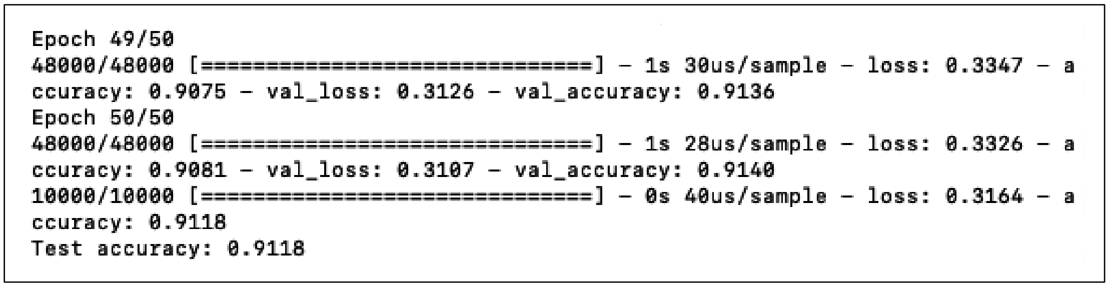
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_layer (Dense)          (None, 10)                7850
=================================================================
Total params: 7,850
Trainable params: 7,850
Non-trainable params: 0
_________________________________________________________________
Train on 48000 samples, validate on 12000 samples
Epoch 1/200
48000/48000 [==============================] - 1s 31us/sample - loss: 2.1276 - a
ccuracy: 0.2322 - val_loss: 1.9508 - val_accuracy: 0.3908
Epoch 2/200
48000/48000 [==============================] - 1s 23us/sample - loss: 1.8251 - a
ccuracy: 0.5141 - val_loss: 1.6848 - val_accuracy: 0.6277
Epoch 3/200
48000/48000 [==============================] - 1s 25us/sample - loss: 1.5992 - a
ccuracy: 0.6531 - val_loss: 1.4838 - val_accuracy: 0.7150
Epoch 4/200
48000/48000 [==============================] - 1s 27us/sample - loss: 1.4281 - a
ccuracy: 0.7115 - val_loss: 1.3304 - val_accuracy: 0.7551
Epoch 5/200
```



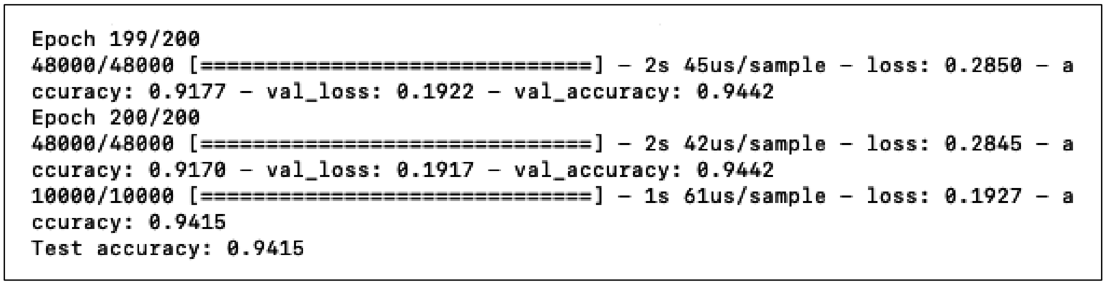
```
Epoch 199/200
48000/48000 [==============================] - 1s 22us/sample - loss: 0.3684 - a
ccuracy: 0.8995 - val_loss: 0.3464 - val_accuracy: 0.9071
Epoch 200/200
48000/48000 [==============================] - 1s 23us/sample - loss: 0.3680 - a
ccuracy: 0.8996 - val_loss: 0.3461 - val_accuracy: 0.9070
10000/10000 [==============================] - 1s 54us/sample - loss: 0.3465 - a
ccuracy: 0.9071
Test accuracy: 0.9071
```



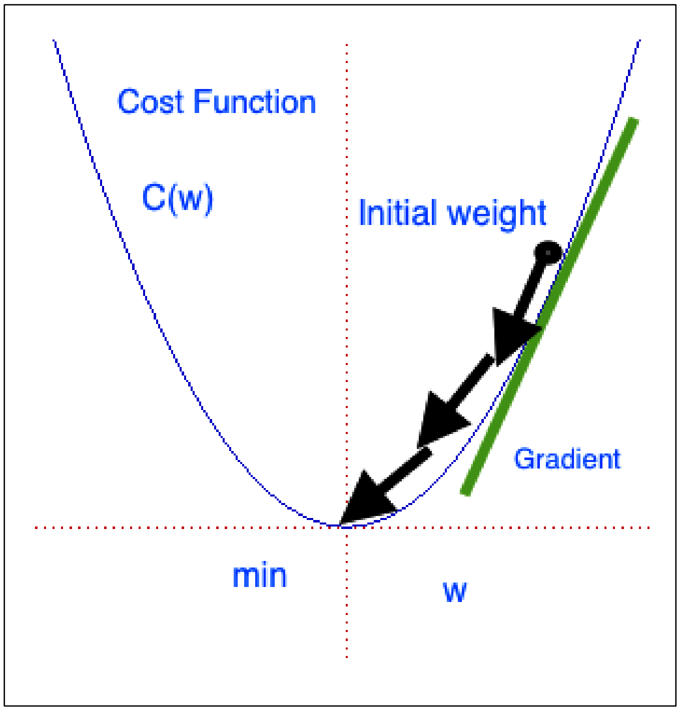
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_layer (Dense)          (None, 128)               100480
_________________________________________________________________
dense_layer_2 (Dense)        (None, 128)               16512
_________________________________________________________________
dense_layer_3 (Dense)        (None, 10)                1290
=================================================================
Total params: 118,282
Trainable params: 118,282
Non-trainable params: 0
_________________________________________________________________
Train on 48000 samples, validate on 12000 samples
Epoch 1/200
48000/48000 [==============================] - 3s 63us/sample - loss: 2.2507 - a
ccuracy: 0.2086 - val_loss: 2.1592 - val_accuracy: 0.3266
```

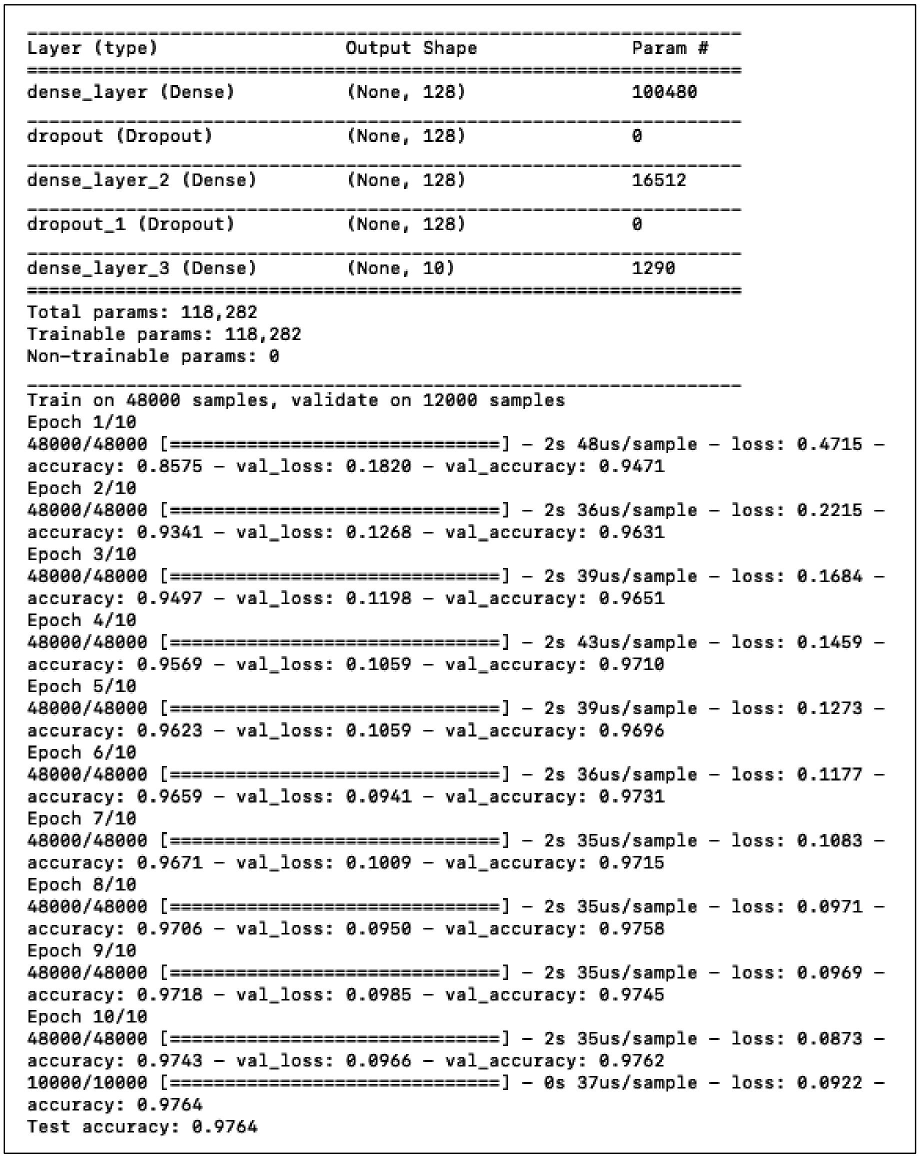


```
Epoch 49/50
48000/48000 [==============================] - 1s 30us/sample - loss: 0.3347 - a
ccuracy: 0.9075 - val_loss: 0.3126 - val_accuracy: 0.9136
Epoch 50/50
48000/48000 [==============================] - 1s 28us/sample - loss: 0.3326 - a
ccuracy: 0.9081 - val_loss: 0.3107 - val_accuracy: 0.9140
10000/10000 [==============================] - 0s 40us/sample - loss: 0.3164 - a
ccuracy: 0.9118
Test accuracy: 0.9118
```

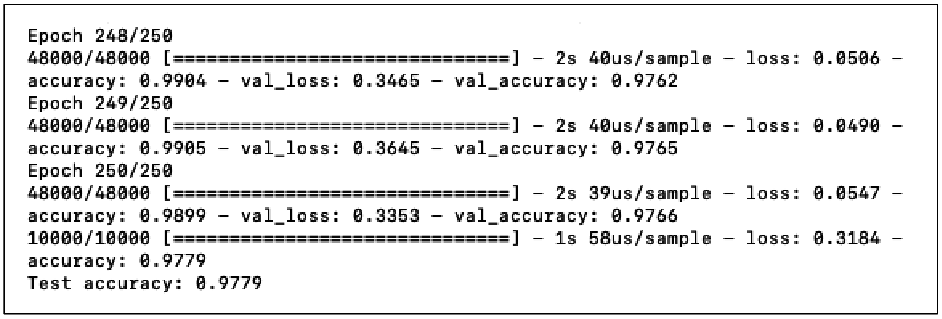


```
Epoch 199/200
48000/48000 [==============================] - 2s 45us/sample - loss: 0.2850 - a
ccuracy: 0.9177 - val_loss: 0.1922 - val_accuracy: 0.9442
Epoch 200/200
48000/48000 [==============================] - 2s 42us/sample - loss: 0.2845 - a
ccuracy: 0.9170 - val_loss: 0.1917 - val_accuracy: 0.9442
10000/10000 [==============================] - 1s 61us/sample - loss: 0.1927 - a
ccuracy: 0.9415
Test accuracy: 0.9415
```

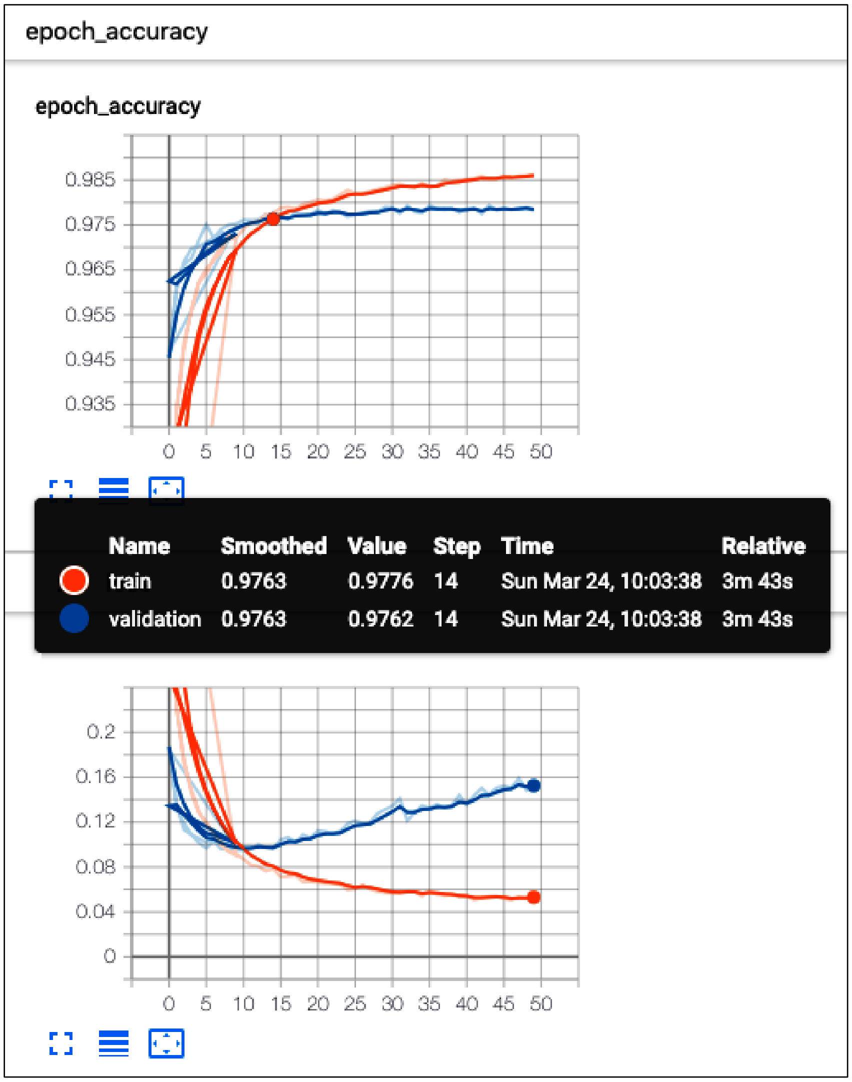

Cost Function
C(w)
Initial weight
Gradient
min
w



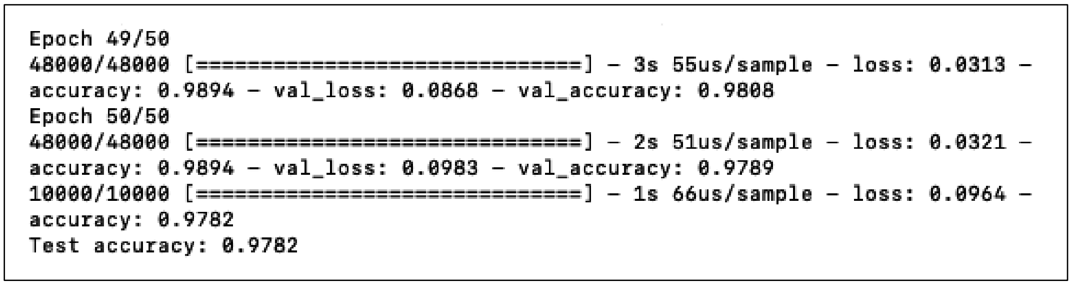
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_layer (Dense)          (None, 128)               100480
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_layer_2 (Dense)        (None, 128)               16512
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_layer_3 (Dense)        (None, 10)                1290
=================================================================
Total params: 118,282
Trainable params: 118,282
Non-trainable params: 0
_________________________________________________________________
Train on 48000 samples, validate on 12000 samples
Epoch 1/10
48000/48000 [==============================] - 2s 48us/sample - loss: 0.4715 -
accuracy: 0.8575 - val_loss: 0.1820 - val_accuracy: 0.9471
Epoch 2/10
48000/48000 [==============================] - 2s 36us/sample - loss: 0.2215 -
accuracy: 0.9341 - val_loss: 0.1268 - val_accuracy: 0.9631
Epoch 3/10
48000/48000 [==============================] - 2s 39us/sample - loss: 0.1684 -
accuracy: 0.9497 - val_loss: 0.1198 - val_accuracy: 0.9651
Epoch 4/10
48000/48000 [==============================] - 2s 43us/sample - loss: 0.1459 -
accuracy: 0.9569 - val_loss: 0.1059 - val_accuracy: 0.9710
Epoch 5/10
48000/48000 [==============================] - 2s 39us/sample - loss: 0.1273 -
accuracy: 0.9623 - val_loss: 0.1059 - val_accuracy: 0.9696
Epoch 6/10
48000/48000 [==============================] - 2s 36us/sample - loss: 0.1177 -
accuracy: 0.9659 - val_loss: 0.0941 - val_accuracy: 0.9731
Epoch 7/10
48000/48000 [==============================] - 2s 35us/sample - loss: 0.1083 -
accuracy: 0.9671 - val_loss: 0.1009 - val_accuracy: 0.9715
Epoch 8/10
48000/48000 [==============================] - 2s 35us/sample - loss: 0.0971 -
accuracy: 0.9706 - val_loss: 0.0950 - val_accuracy: 0.9758
Epoch 9/10
48000/48000 [==============================] - 2s 35us/sample - loss: 0.0969 -
accuracy: 0.9718 - val_loss: 0.0985 - val_accuracy: 0.9745
Epoch 10/10
48000/48000 [==============================] - 2s 35us/sample - loss: 0.0873 -
accuracy: 0.9743 - val_loss: 0.0966 - val_accuracy: 0.9762
10000/10000 [==============================] - 0s 37us/sample - loss: 0.0922 -
accuracy: 0.9764
Test accuracy: 0.9764
```



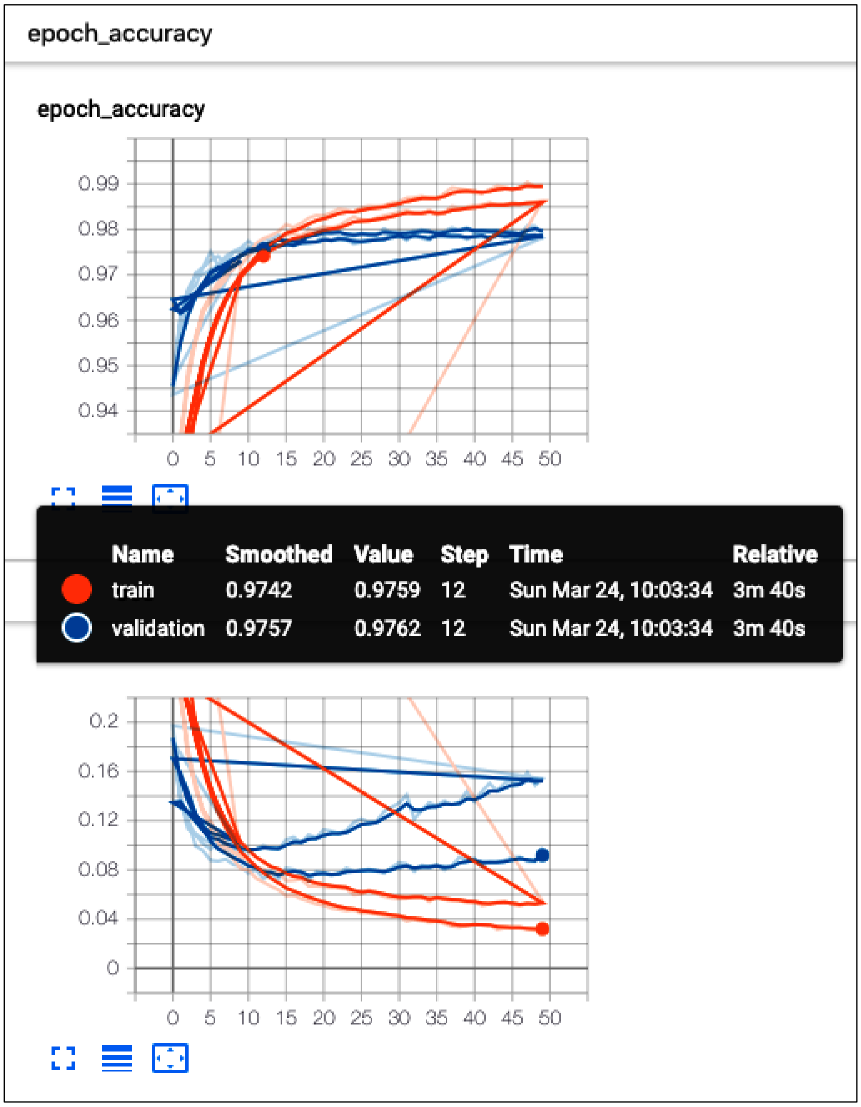
```
Epoch 248/250
48000/48000 [==============================] - 2s 40us/sample - loss: 0.0506 -
accuracy: 0.9904 - val_loss: 0.3465 - val_accuracy: 0.9762
Epoch 249/250
48000/48000 [==============================] - 2s 40us/sample - loss: 0.0490 -
accuracy: 0.9905 - val_loss: 0.3645 - val_accuracy: 0.9765
Epoch 250/250
48000/48000 [==============================] - 2s 39us/sample - loss: 0.0547 -
accuracy: 0.9899 - val_loss: 0.3353 - val_accuracy: 0.9766
10000/10000 [==============================] - 1s 58us/sample - loss: 0.3184 -
accuracy: 0.9779
Test accuracy: 0.9779
```



	Name	Smoothed	Value	Step	Time	Relative
●	train	0.9763	0.9776	14	Sun Mar 24, 10:03:38	3m 43s
●	validation	0.9763	0.9762	14	Sun Mar 24, 10:03:38	3m 43s



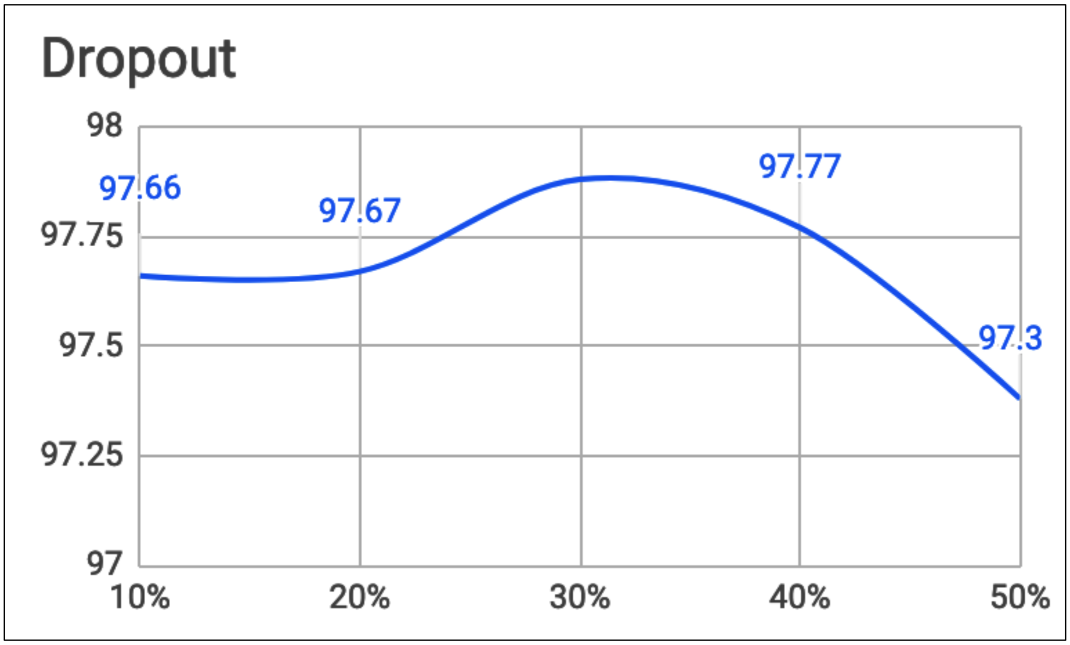
```
Epoch 49/50
48000/48000 [==============================] - 3s 55us/sample - loss: 0.0313 -
accuracy: 0.9894 - val_loss: 0.0868 - val_accuracy: 0.9808
Epoch 50/50
48000/48000 [==============================] - 2s 51us/sample - loss: 0.0321 -
accuracy: 0.9894 - val_loss: 0.0983 - val_accuracy: 0.9789
10000/10000 [==============================] - 1s 66us/sample - loss: 0.0964 -
accuracy: 0.9782
Test accuracy: 0.9782
```

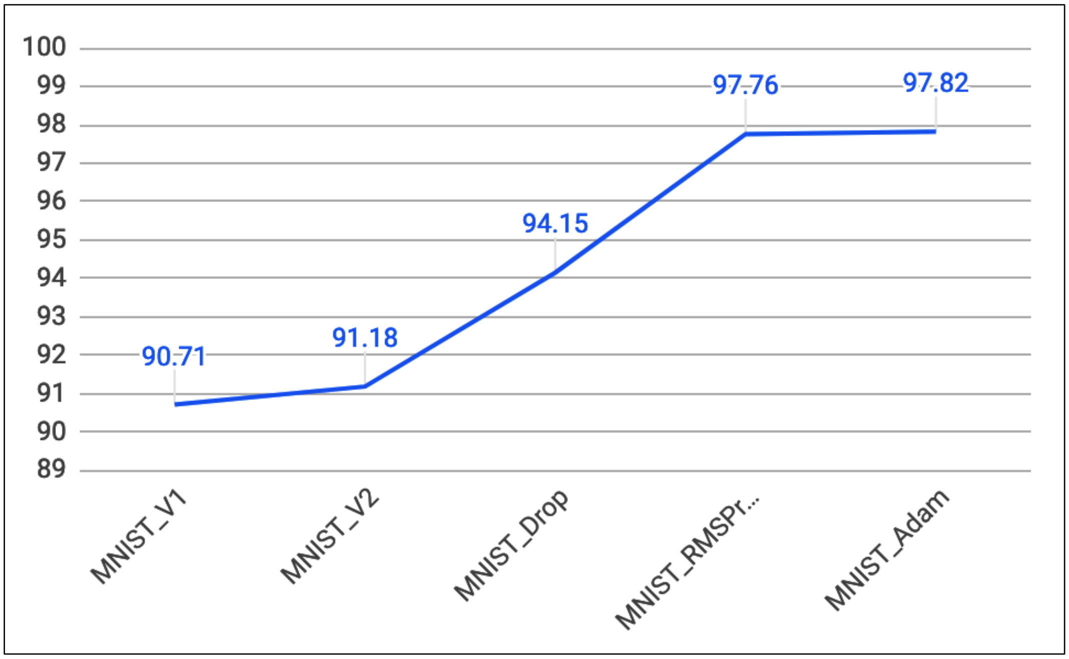


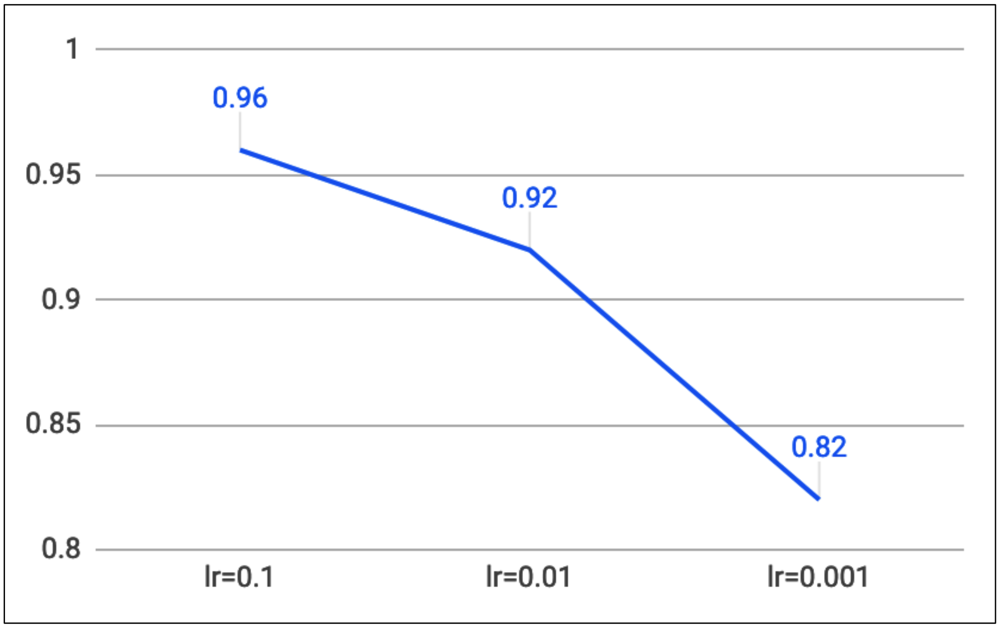
epoch_accuracy

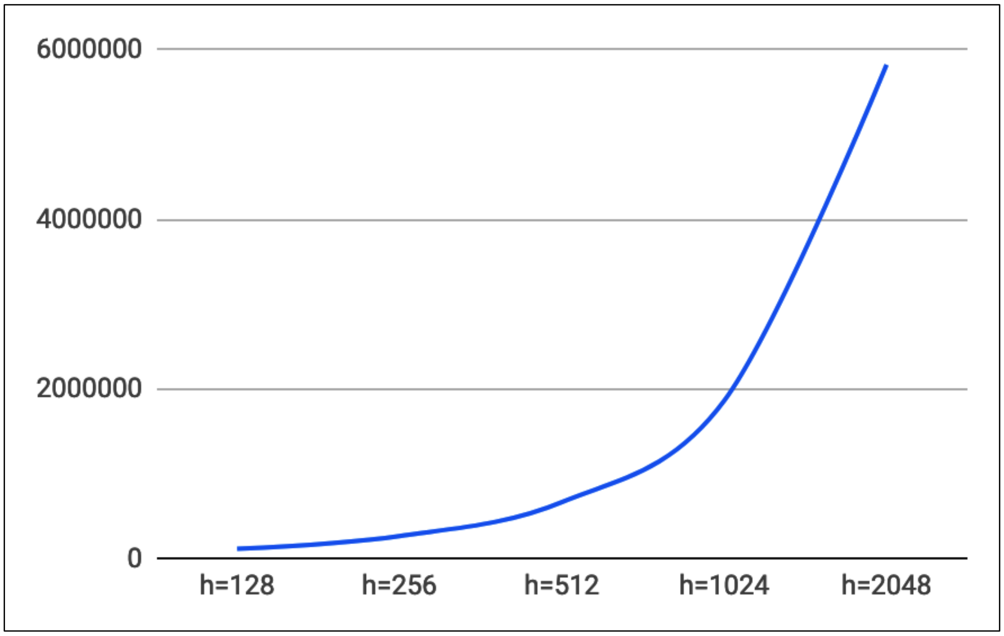
epoch_accuracy

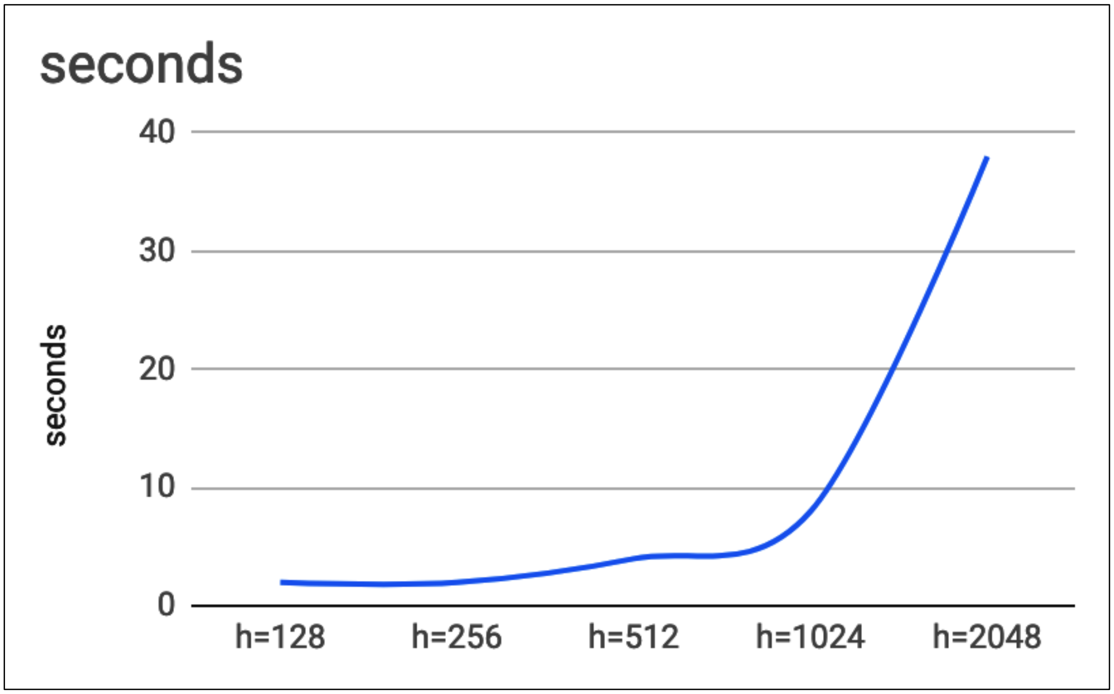
	Name	Smoothed	Value	Step	Time	Relative
●	train	0.9742	0.9759	12	Sun Mar 24, 10:03:34	3m 40s
○	validation	0.9757	0.9762	12	Sun Mar 24, 10:03:34	3m 40s

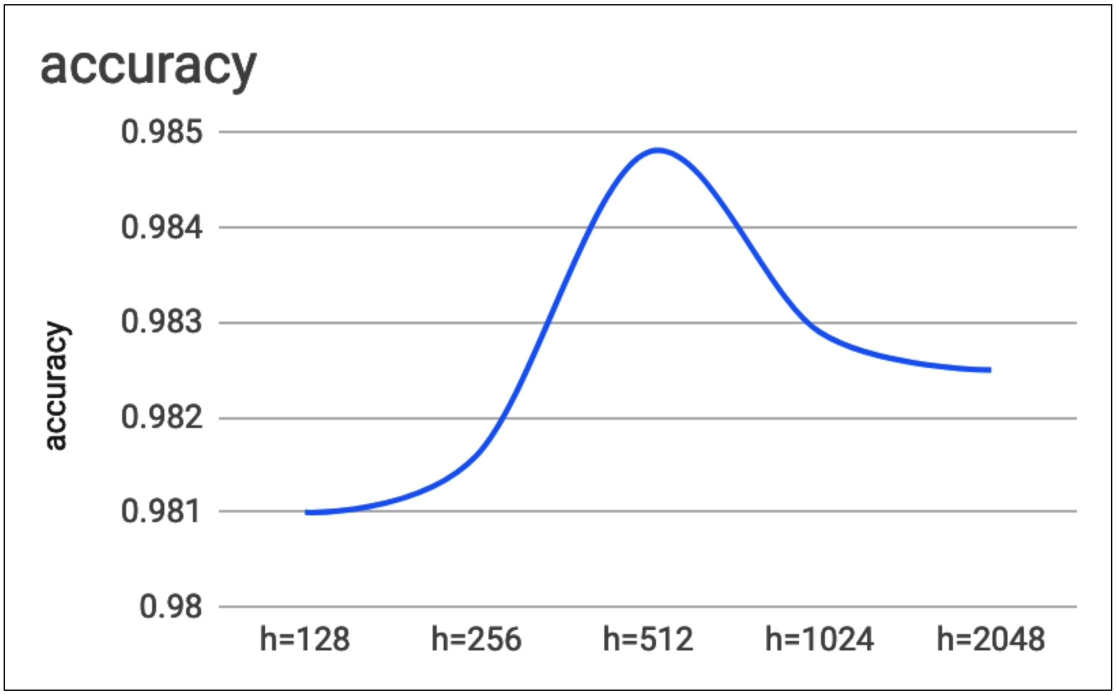

Dropout
98
97.75
97.5
97.25
97
10%
20%
30%
40%
50%
97.66
97.67
97.77
97.3

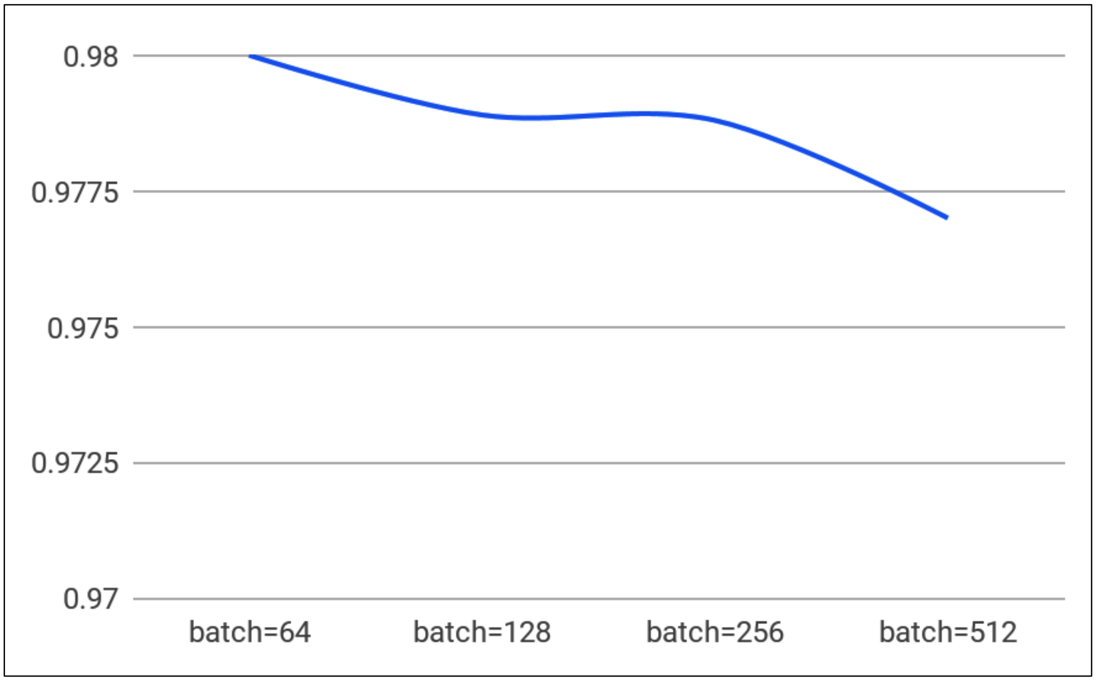

100
99
98
97
96
95
94
93
92
91
90
89
90.71
91.18
94.15
97.76
97.82
MNIST_V1
MNIST_V2
MNIST_Drop
MNIST_RMSPr...
MNIST_Adam

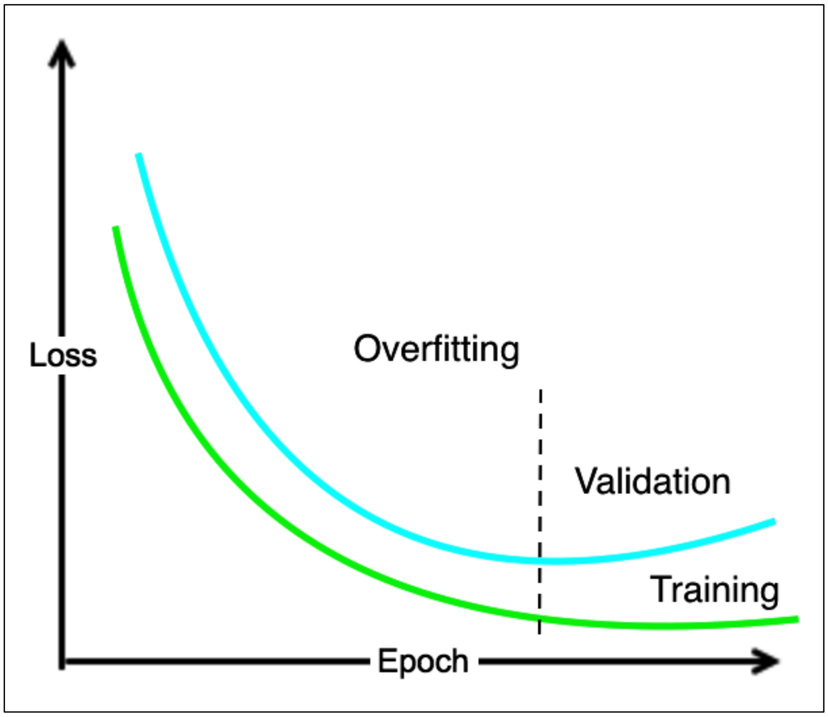

1
0.95
0.9
0.85
0.8
0.96
0.92
0.82
lr=0.1
lr=0.01
lr=0.001

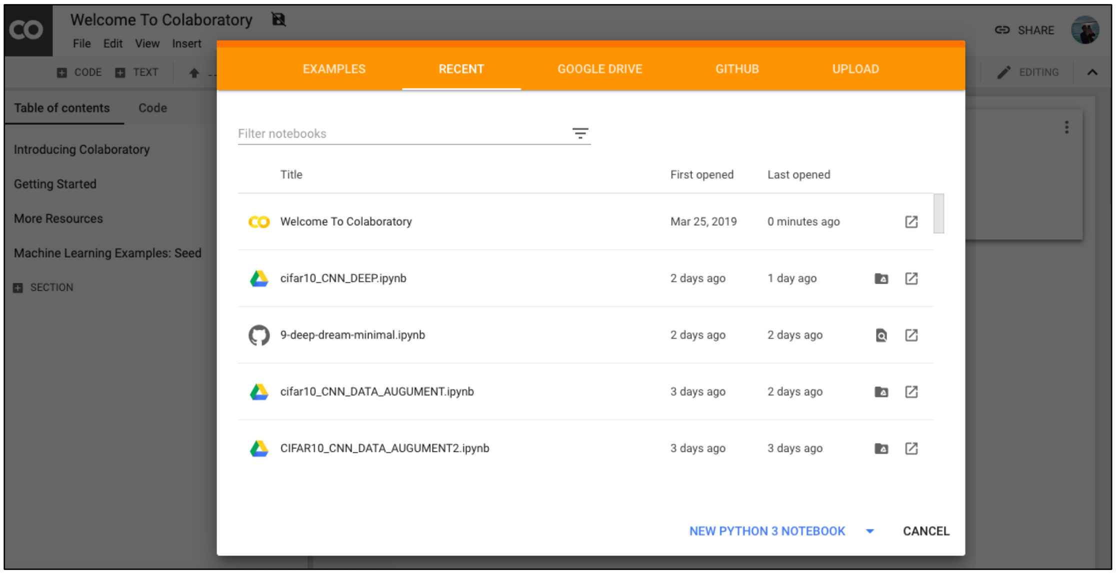

6000000
4000000
2000000
0
h=128
h=256
h=512
h=1024
h=2048


seconds
40
30
20
10
0
seconds
h=128
h=256
h=512
h=1024
h=2048

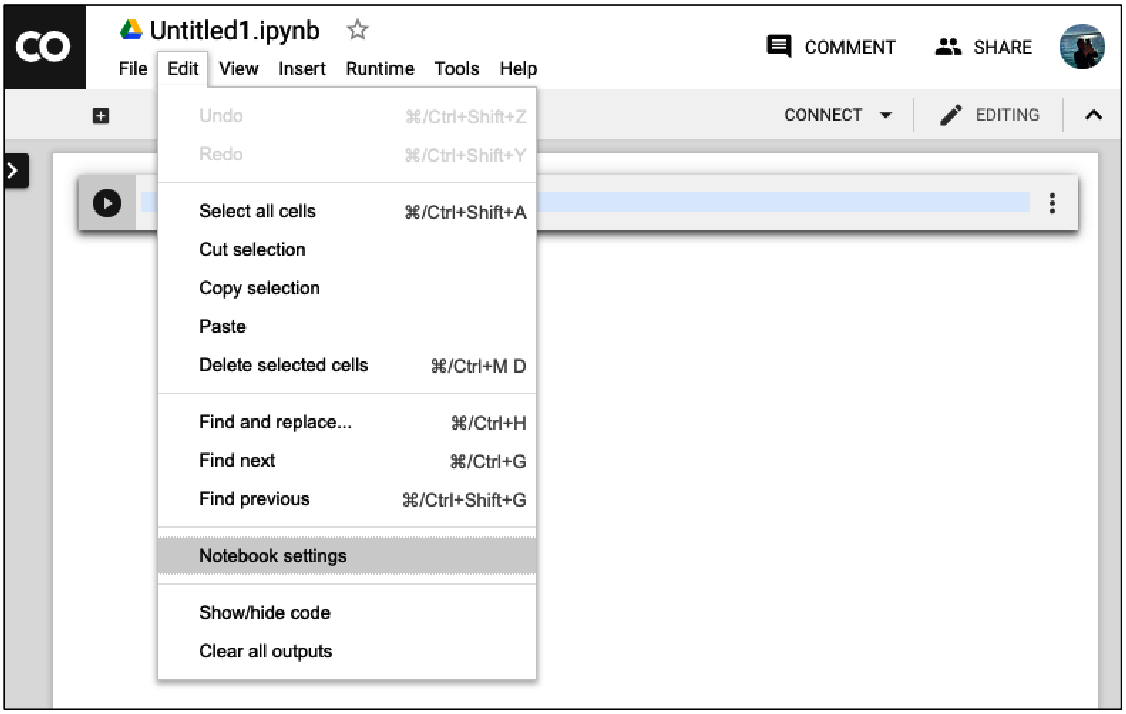

accuracy
0.985
0.984
0.983
0.982
0.981
0.98
accuracy
h=128
h=256
h=512
h=1024
h=2048

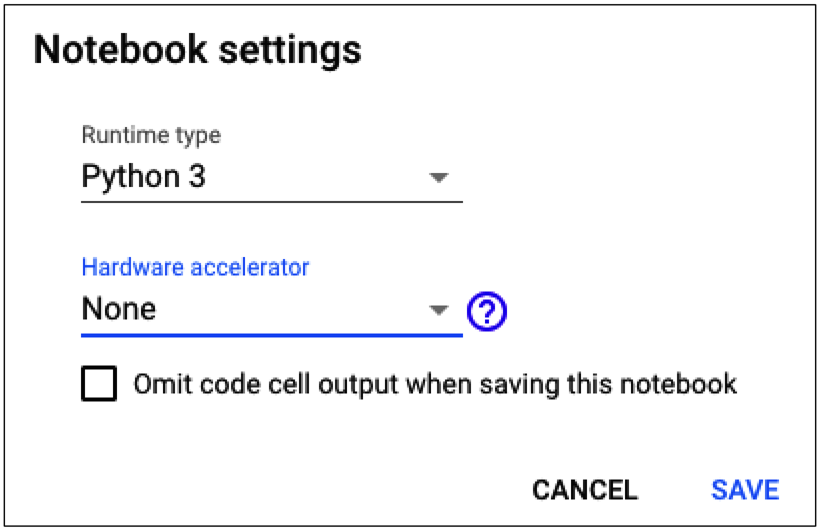

0.98
0.9775
0.975
0.9725
0.97
batch=64
batch=128
batch=256
batch=512


Loss
Overfitting
Validation
Training
Epoch



Title	First opened	Last opened
Welcome To Colaboratory	Mar 25, 2019	0 minutes ago
cifar10_CNN_DEEP.ipynb	2 days ago	1 day ago
9-deep-dream-minimal.ipynb	2 days ago	2 days ago
cifar10_CNN_DATA_AUGUMENT.ipynb	3 days ago	2 days ago
CIFAR10_CNN_DATA_AUGUMENT2.ipynb	3 days ago	3 days ago



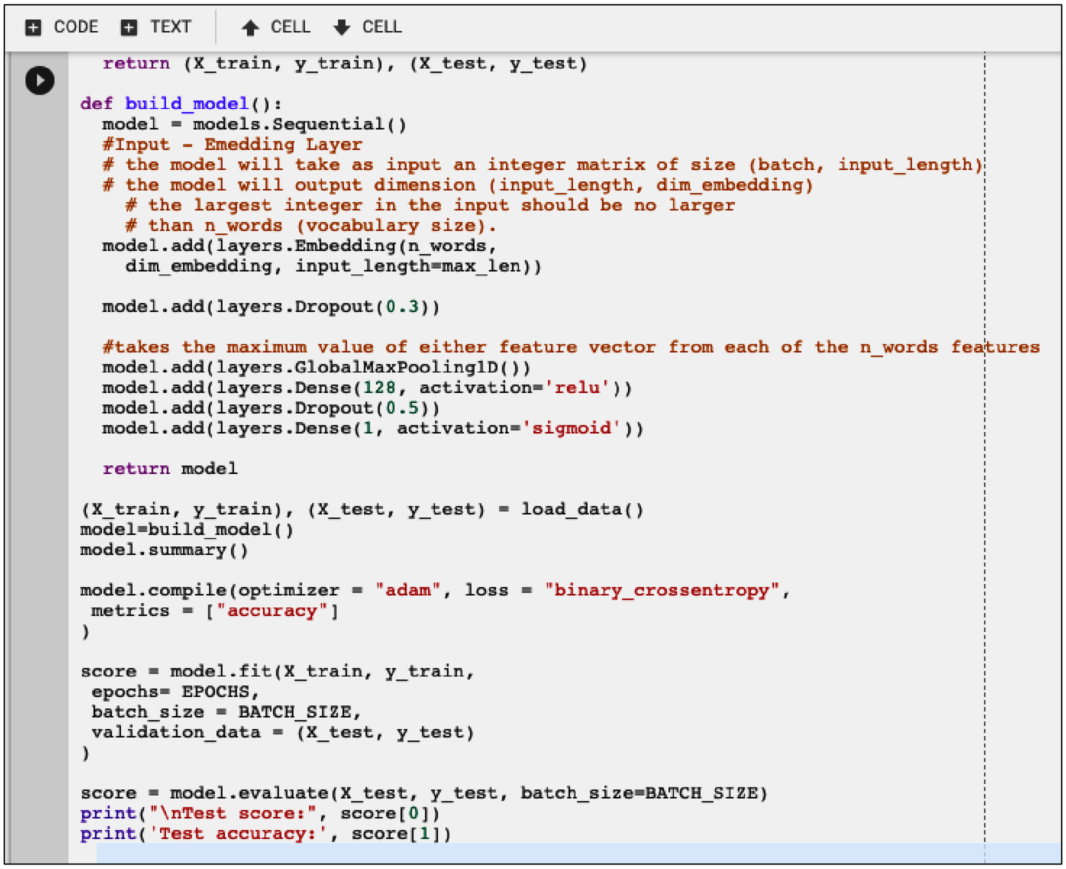


Notebook settings

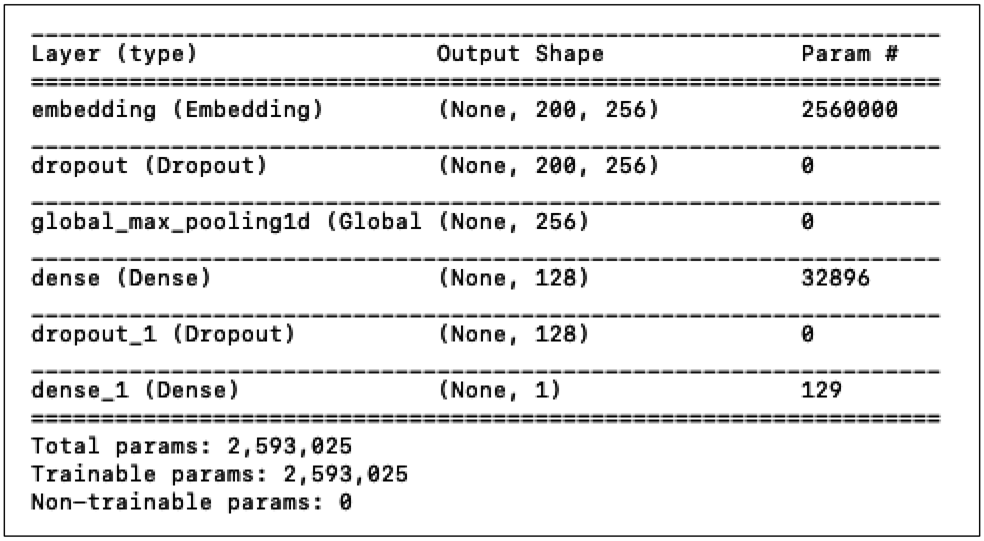
Runtime type
Python 3

Hardware accelerator
None

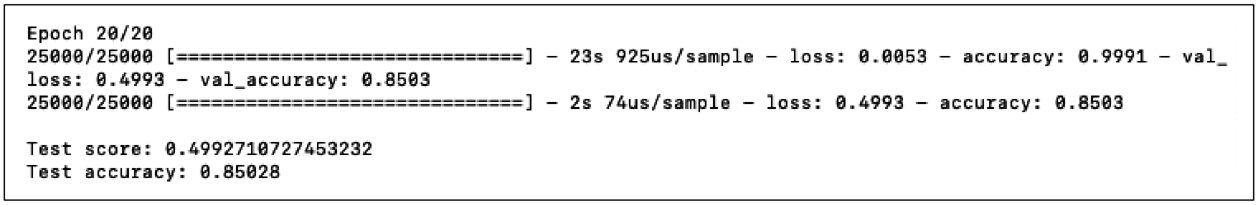
□ Omit code cell output when saving this notebook



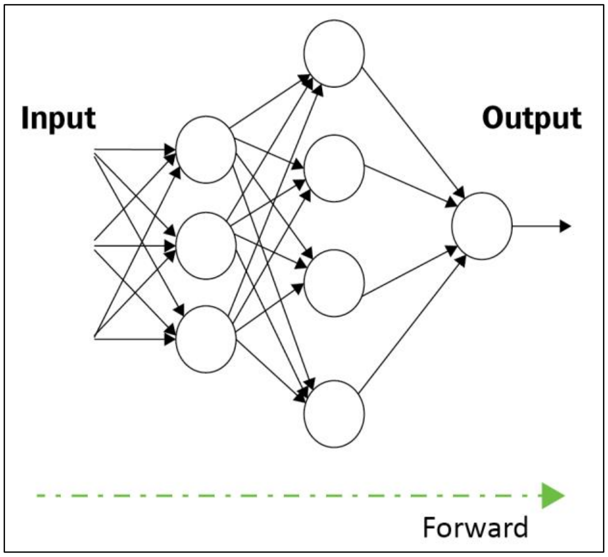
```
  return (X_train, y_train), (X_test, y_test)

def build_model():
  model = models.Sequential()
  #Input - Emedding Layer
  # the model will take as input an integer matrix of size (batch, input_length)
  # the model will output dimension (input_length, dim_embedding)
    # the largest integer in the input should be no larger
    # than n_words (vocabulary size).
  model.add(layers.Embedding(n_words,
    dim_embedding, input_length=max_len))

  model.add(layers.Dropout(0.3))

  #takes the maximum value of either feature vector from each of the n_words features
  model.add(layers.GlobalMaxPooling1D())
  model.add(layers.Dense(128, activation='relu'))
  model.add(layers.Dropout(0.5))
  model.add(layers.Dense(1, activation='sigmoid'))

  return model

(X_train, y_train), (X_test, y_test) = load_data()
model=build_model()
model.summary()

model.compile(optimizer = "adam", loss = "binary_crossentropy",
 metrics = ["accuracy"]
)

score = model.fit(X_train, y_train,
 epochs= EPOCHS,
 batch_size = BATCH_SIZE,
 validation_data = (X_test, y_test)
)

score = model.evaluate(X_test, y_test, batch_size=BATCH_SIZE)
print("\nTest score:", score[0])
print('Test accuracy:', score[1])
```

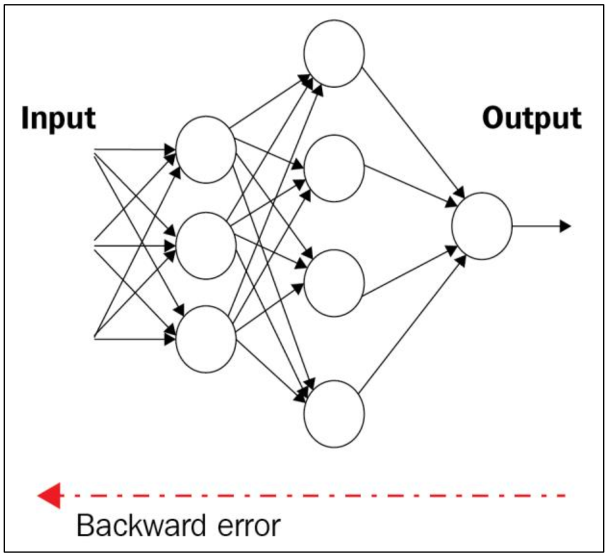


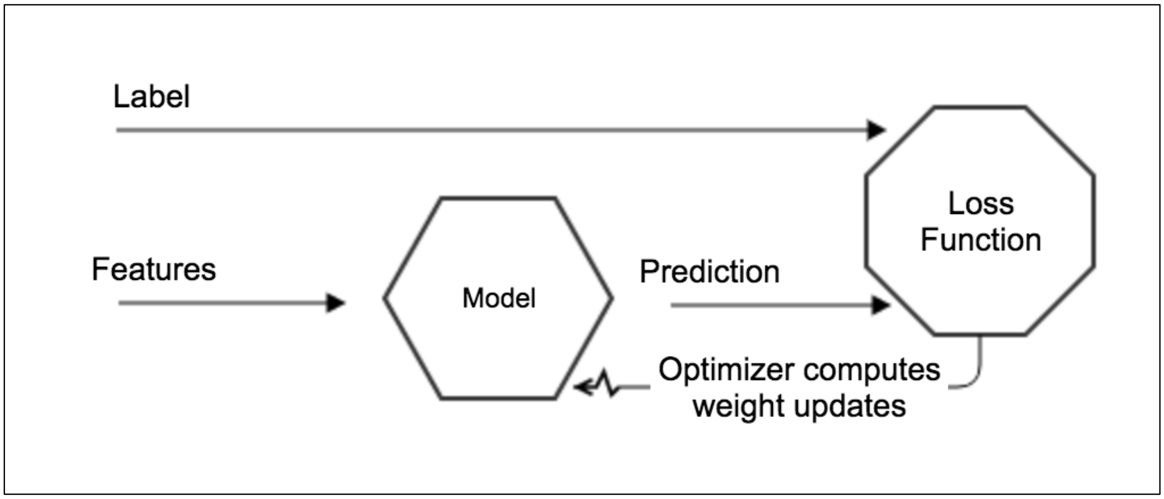
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 200, 256)          2560000
_________________________________________________________________
dropout (Dropout)            (None, 200, 256)          0
_________________________________________________________________
global_max_pooling1d (Global (None, 256)               0
_________________________________________________________________
dense (Dense)                (None, 128)               32896
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 129
=================================================================
Total params: 2,593,025
Trainable params: 2,593,025
Non-trainable params: 0
```



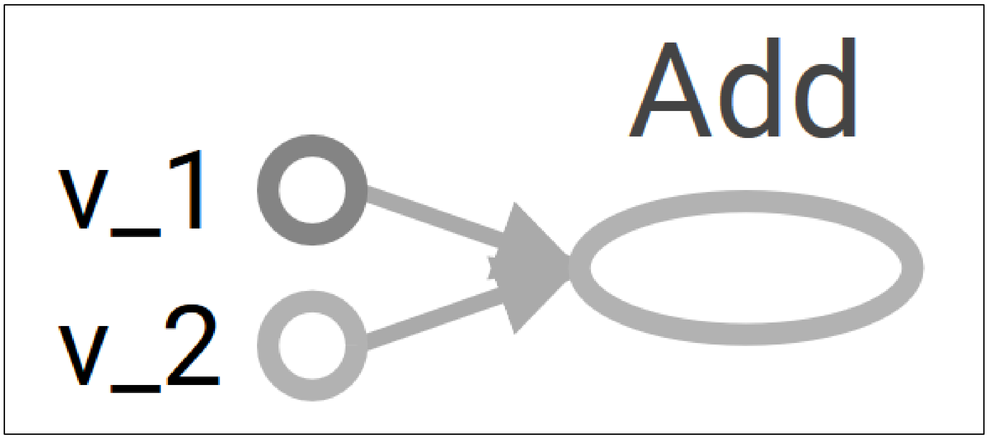
```
Epoch 20/20
25000/25000 [==============================] - 23s 925us/sample - loss: 0.0053 - accuracy: 0.9991 - val_
loss: 0.4993 - val_accuracy: 0.8503
25000/25000 [==============================] - 2s 74us/sample - loss: 0.4993 - accuracy: 0.8503

Test score: 0.4992710727453232
Test accuracy: 0.85028
```

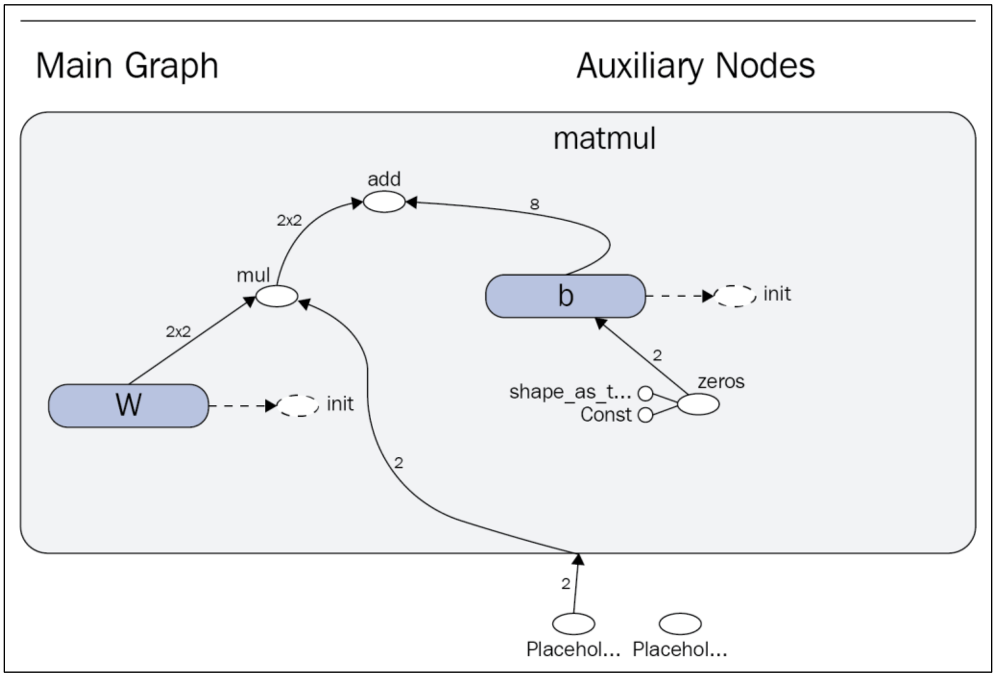


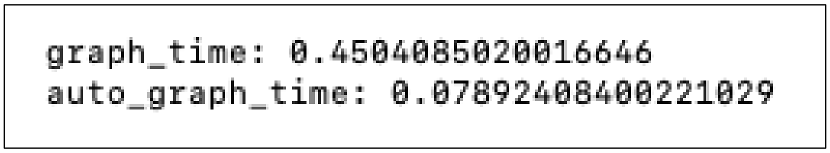

Input
Output
Backward error


Label
Loss Function
Features
Model
Prediction
Optimizer computes weight updates

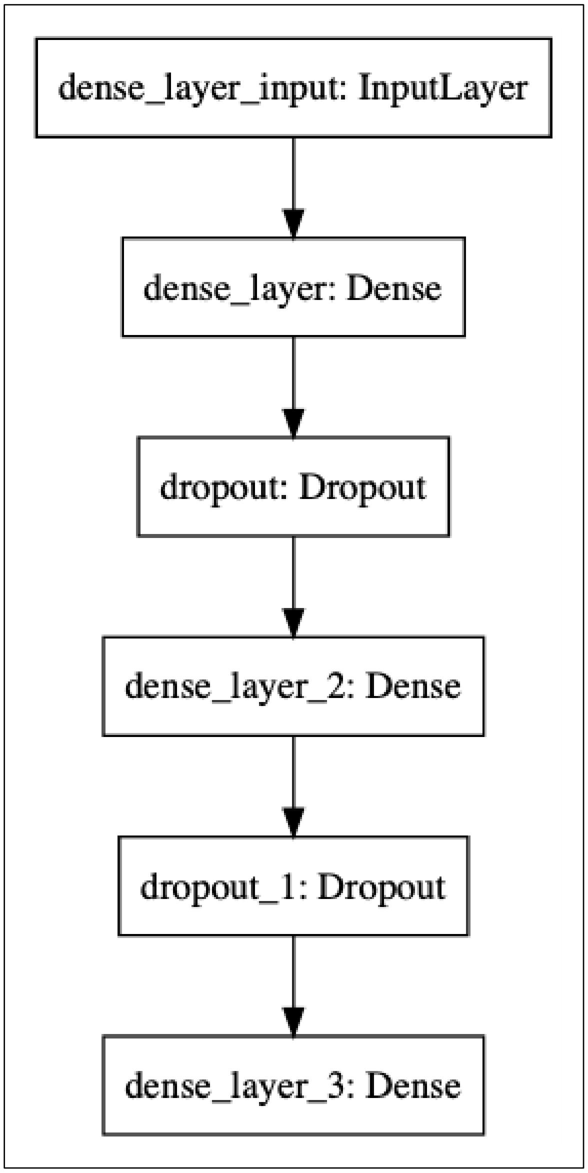
Chapter 2: TensorFlow 1.x and 2.x

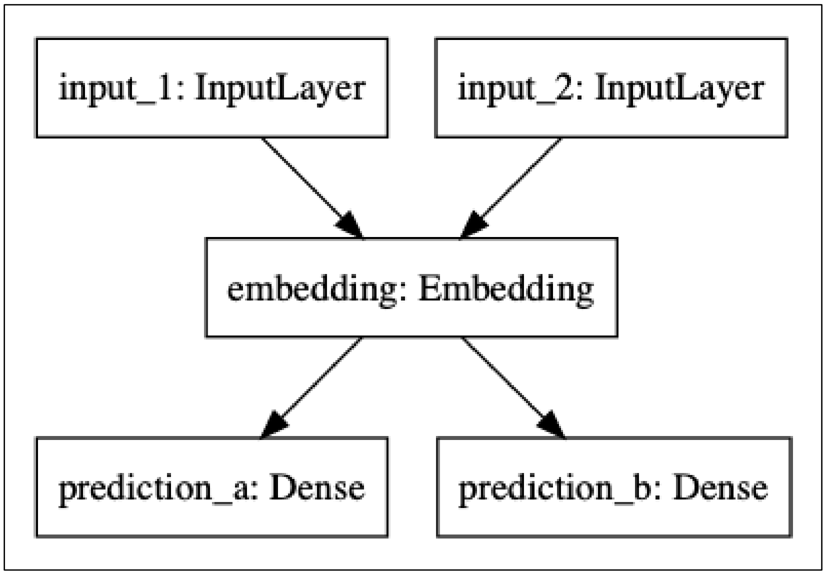


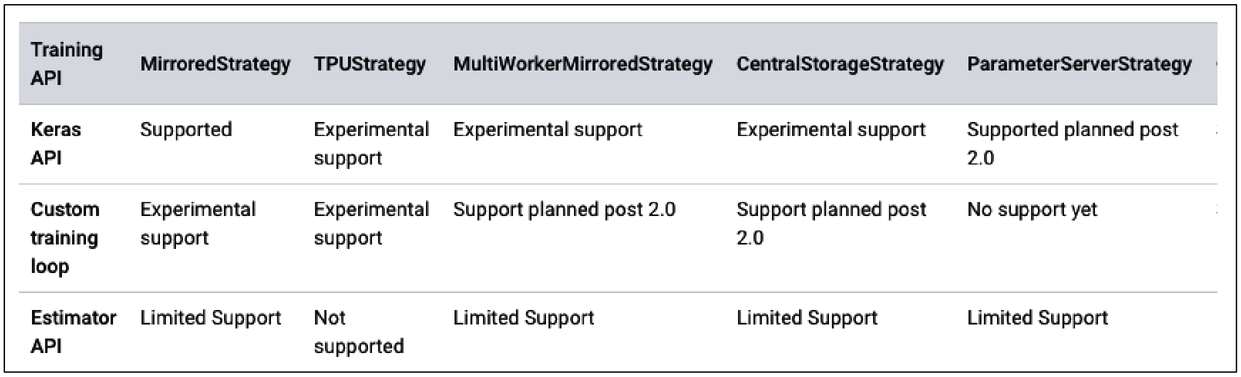




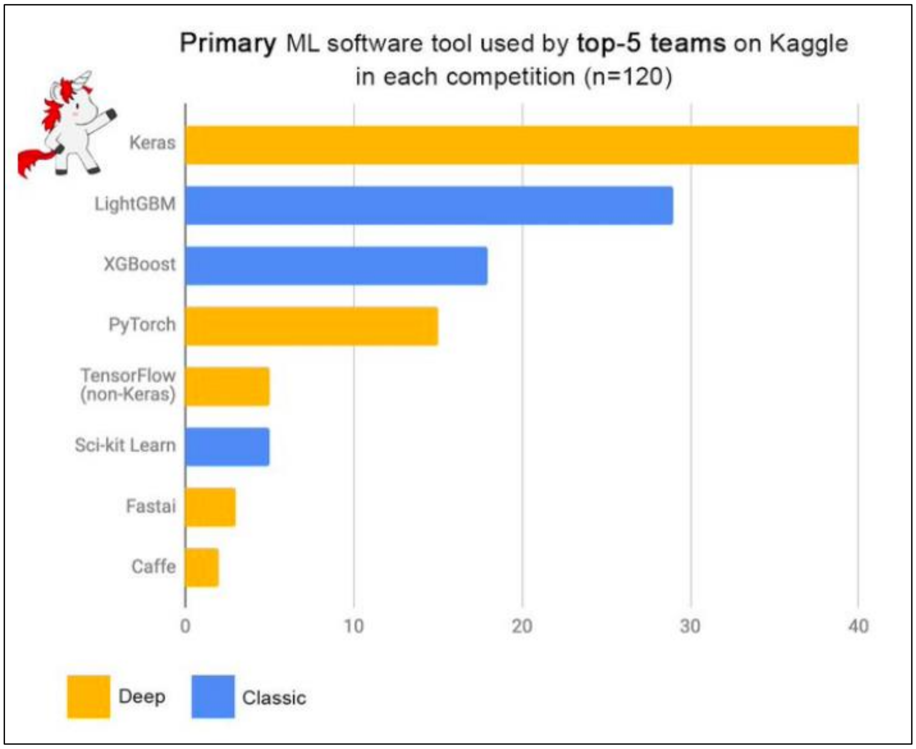
```
graph_time: 0.4504085020016646
auto_graph_time: 0.07892408400221029
```

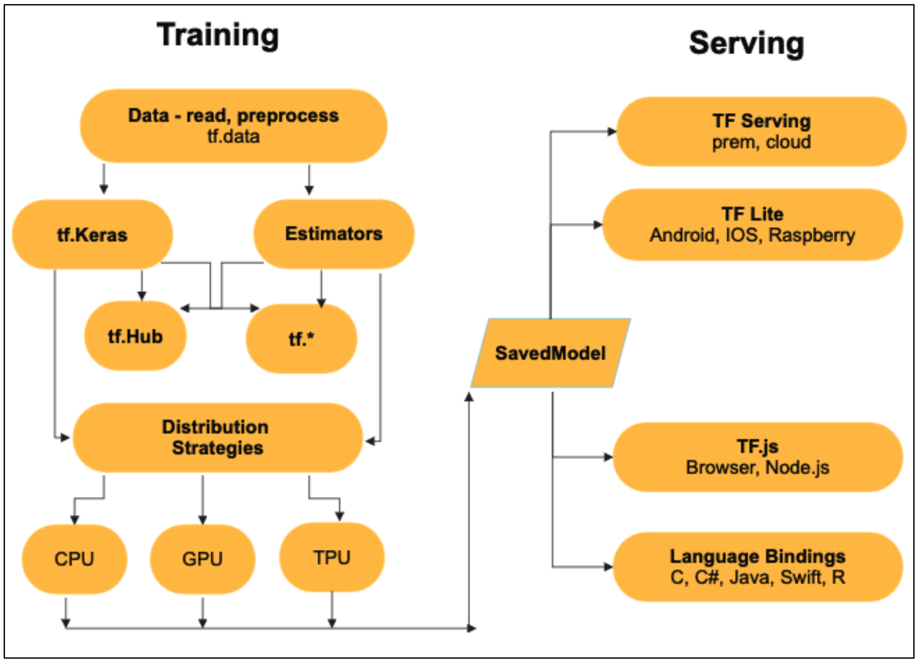

dense_layer_input: InputLayer
dense_layer: Dense
dropout: Dropout
dense_layer_2: Dense
dropout_1: Dropout
dense_layer_3: Dense

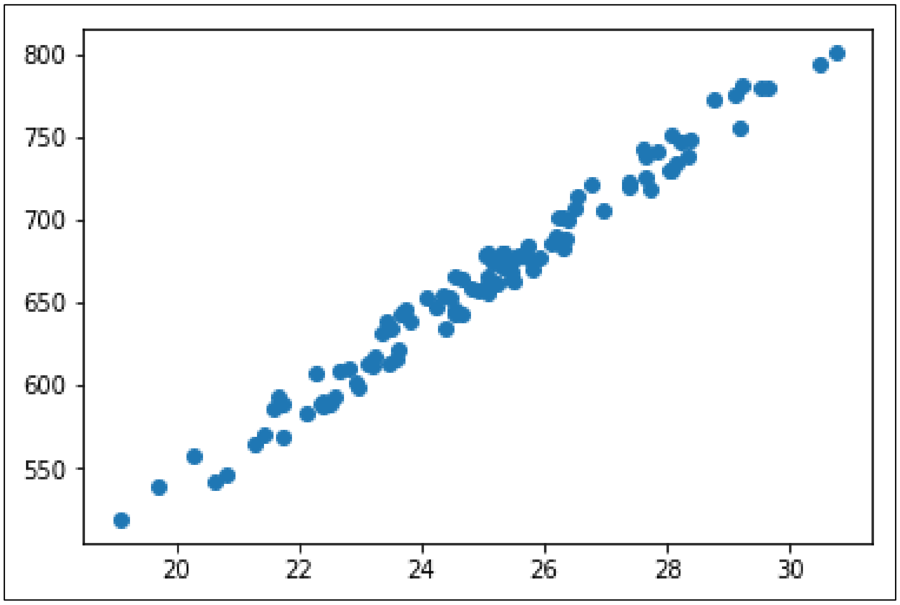


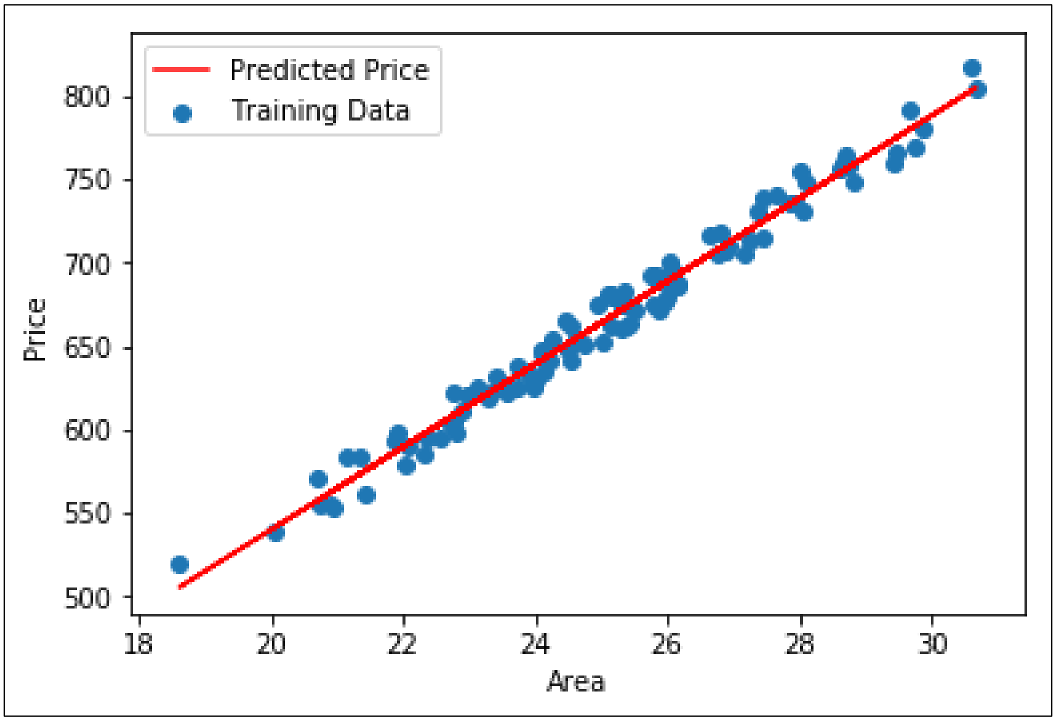


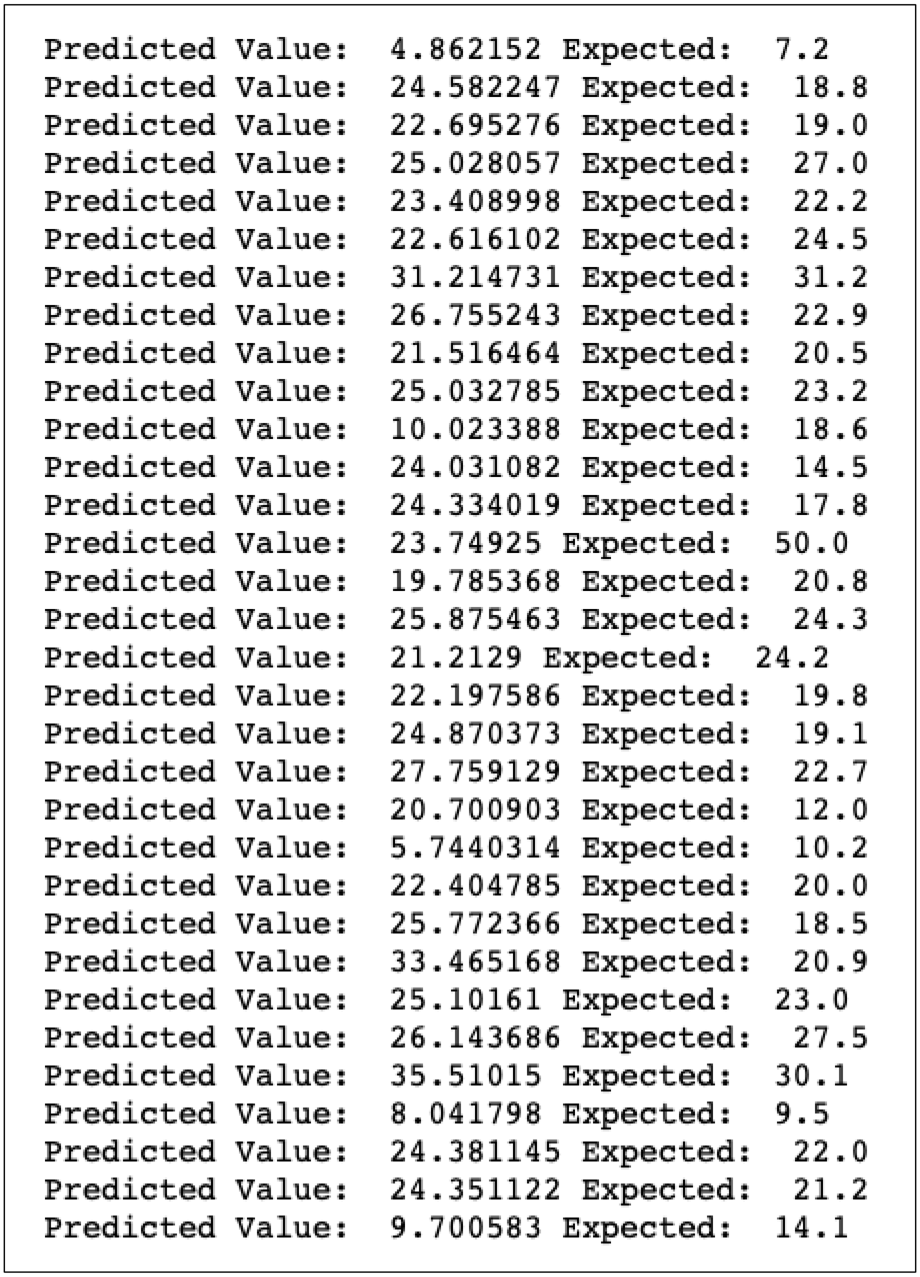
Training API	MirroredStrategy	TPUStrategy	MultiWorkerMirroredStrategy	CentralStorageStrategy	ParameterServerStrategy
Keras API	Supported	Experimental support	Experimental support	Experimental support	Supported planned post 2.0
Custom training loop	Experimental support	Experimental support	Support planned post 2.0	Support planned post 2.0	No support yet
Estimator API	Limited Support	Not supported	Limited Support	Limited Support	Limited Support


Primary ML software tool used by top-5 teams on Kaggle in each competition (n=120)
Keras
LightGBM
XGBoost
PyTorch
TensorFlow (non-Keras)
Sci-kit Learn
Fastai
Caffe
0
10
20
30
40
Deep
Classic

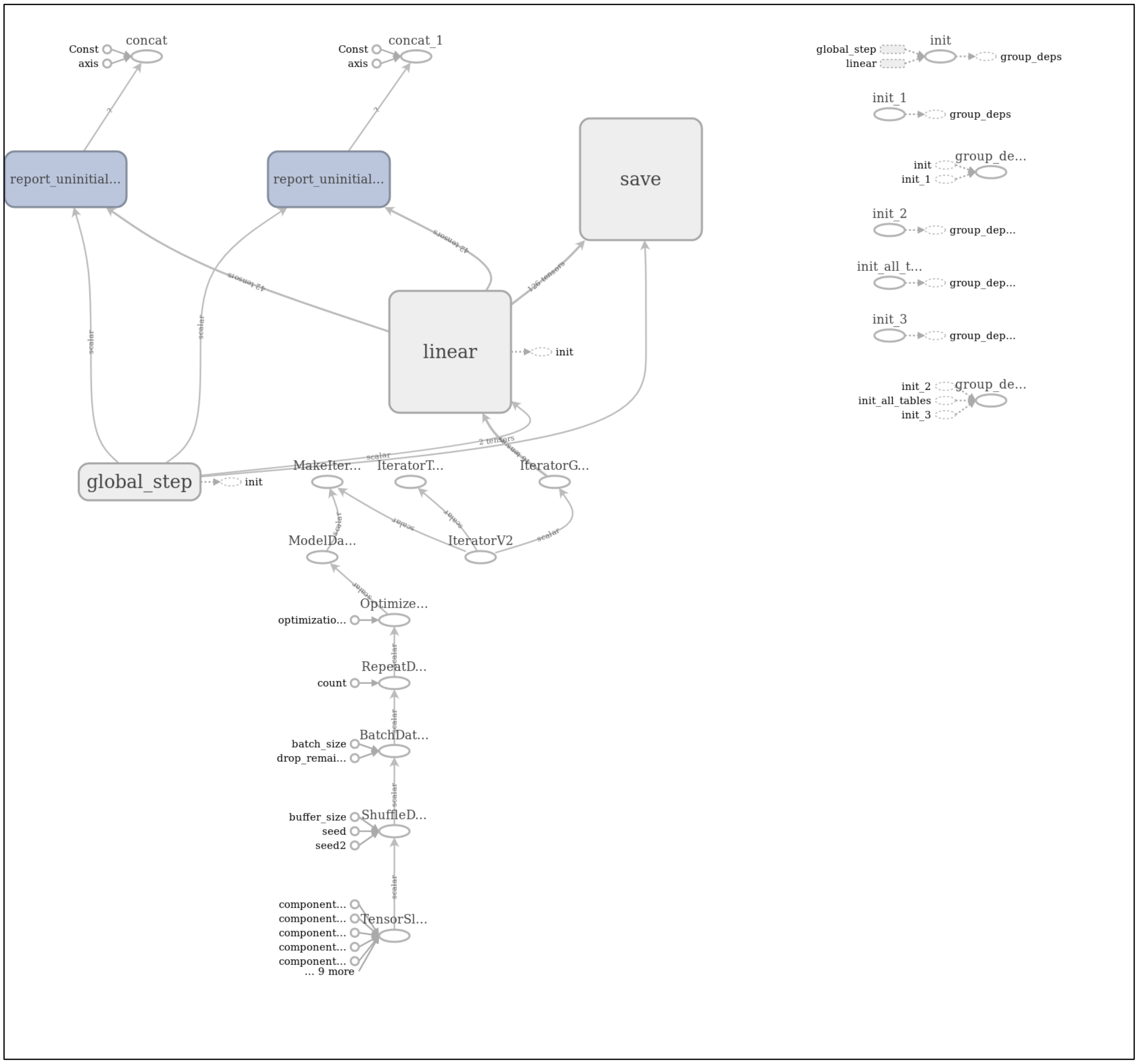

Training
Serving
Data - read, preprocess
tf.data
tf.Keras
Estimators
tf.Hub
tf.*
Distribution Strategies
CPU
GPU
TPU
SavedModel
TF Serving
prem, cloud
TF Lite
Android, IOS, Raspberry
TF.js
Browser, Node.js
Language Bindings
C, C#, Java, Swift, R

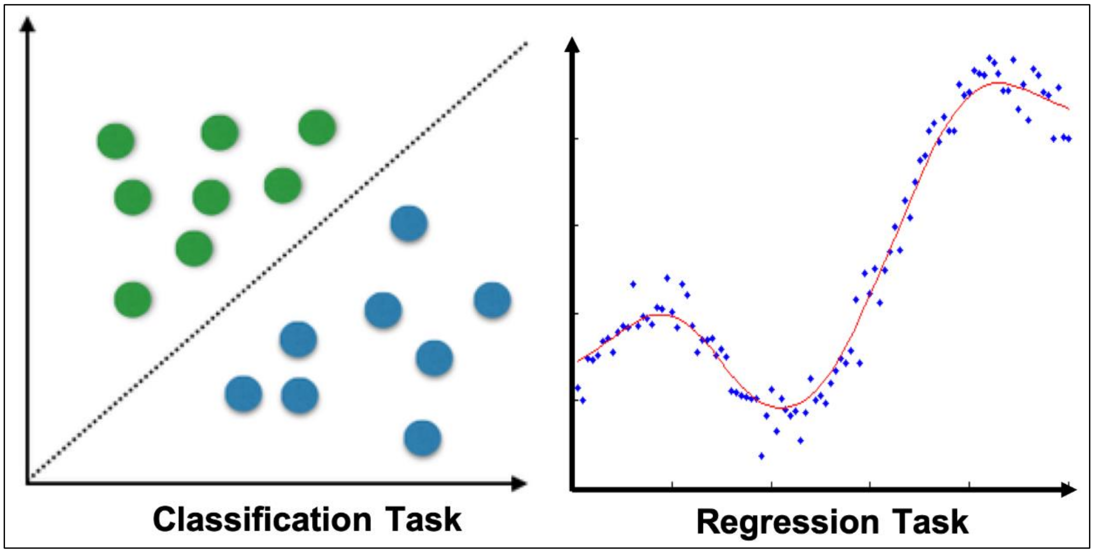

800
750
700
650
600
550
20
22
24
26
28
30

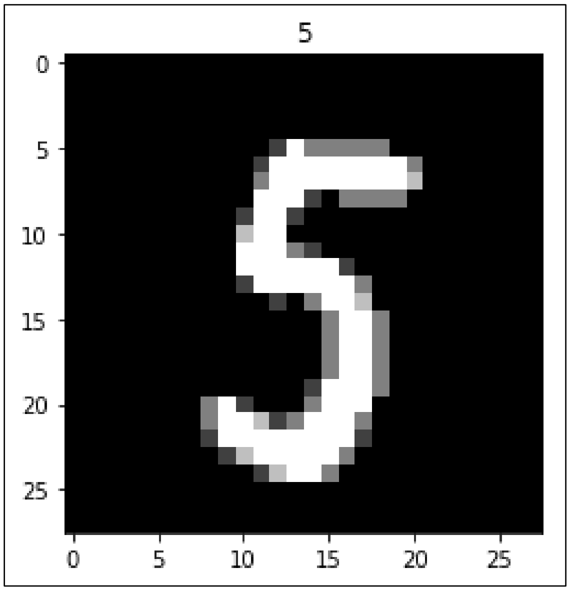

Predicted Price
Training Data
Price
800
750
700
650
600
550
500
18
20
22
24
26
28
30
Area

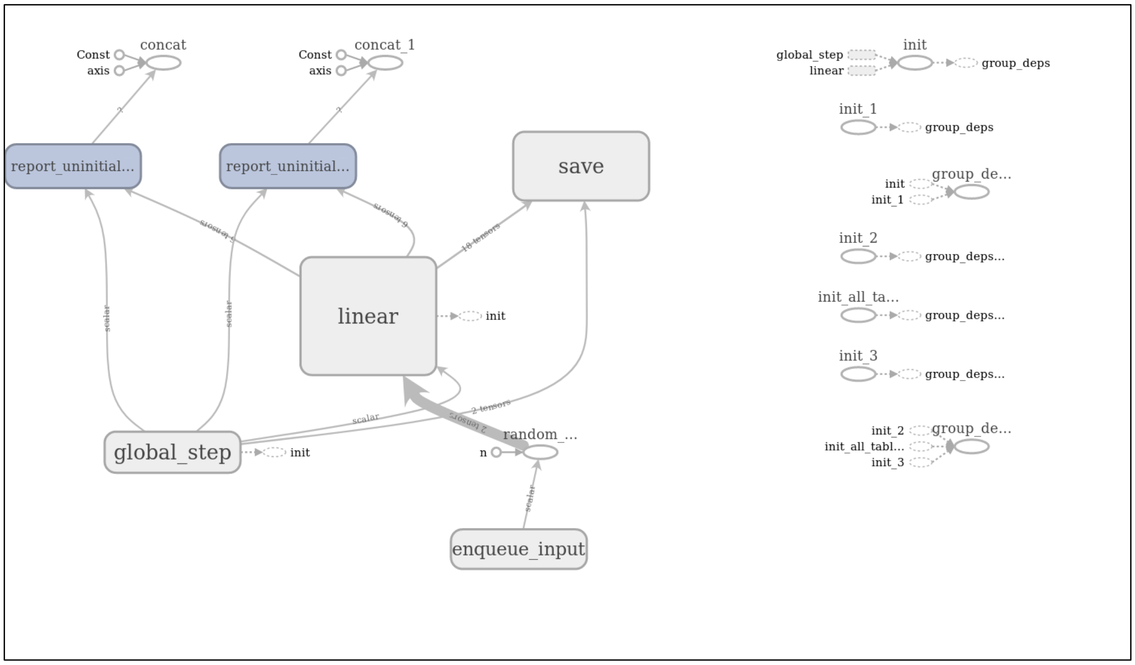


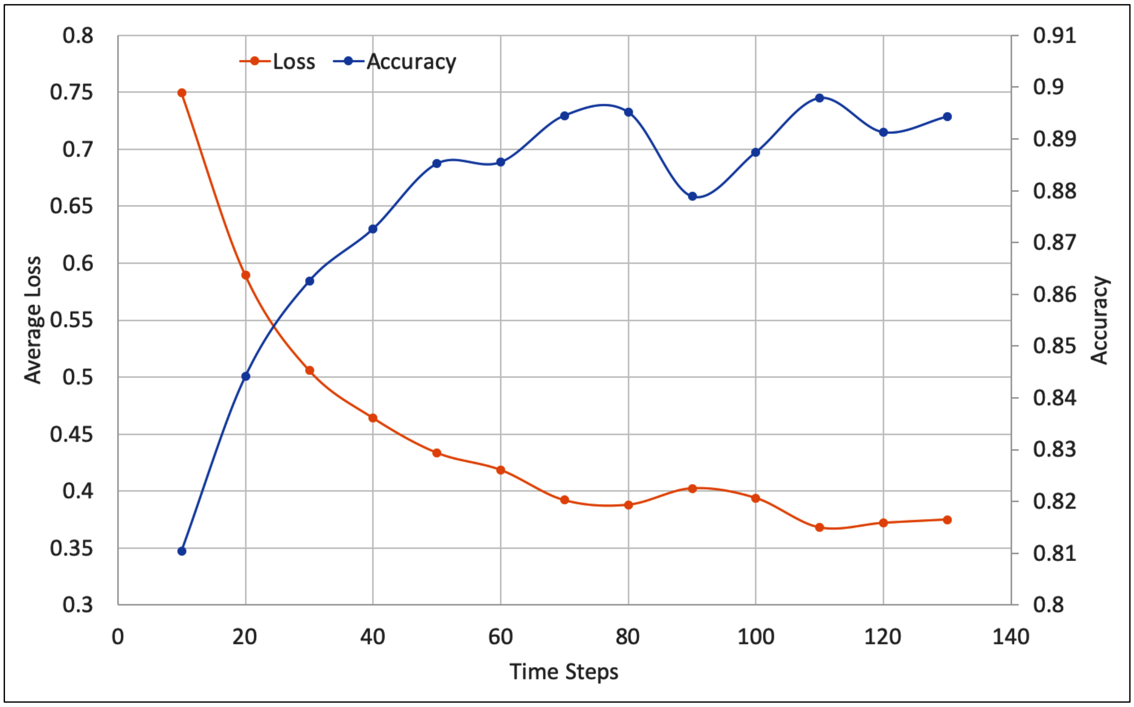
```
Predicted Value:  4.862152 Expected:  7.2
Predicted Value:  24.582247 Expected:  18.8
Predicted Value:  22.695276 Expected:  19.0
Predicted Value:  25.028057 Expected:  27.0
Predicted Value:  23.408998 Expected:  22.2
Predicted Value:  22.616102 Expected:  24.5
Predicted Value:  31.214731 Expected:  31.2
Predicted Value:  26.755243 Expected:  22.9
Predicted Value:  21.516464 Expected:  20.5
Predicted Value:  25.032785 Expected:  23.2
Predicted Value:  10.023388 Expected:  18.6
Predicted Value:  24.031082 Expected:  14.5
Predicted Value:  24.334019 Expected:  17.8
Predicted Value:  23.74925 Expected:  50.0
Predicted Value:  19.785368 Expected:  20.8
Predicted Value:  25.875463 Expected:  24.3
Predicted Value:  21.2129 Expected:  24.2
Predicted Value:  22.197586 Expected:  19.8
Predicted Value:  24.870373 Expected:  19.1
Predicted Value:  27.759129 Expected:  22.7
Predicted Value:  20.700903 Expected:  12.0
Predicted Value:  5.7440314 Expected:  10.2
Predicted Value:  22.404785 Expected:  20.0
Predicted Value:  25.772366 Expected:  18.5
Predicted Value:  33.465168 Expected:  20.9
Predicted Value:  25.10161 Expected:  23.0
Predicted Value:  26.143686 Expected:  27.5
Predicted Value:  35.51015 Expected:  30.1
Predicted Value:  8.041798 Expected:  9.5
Predicted Value:  24.381145 Expected:  22.0
Predicted Value:  24.351122 Expected:  21.2
Predicted Value:  9.700583 Expected:  14.1
```

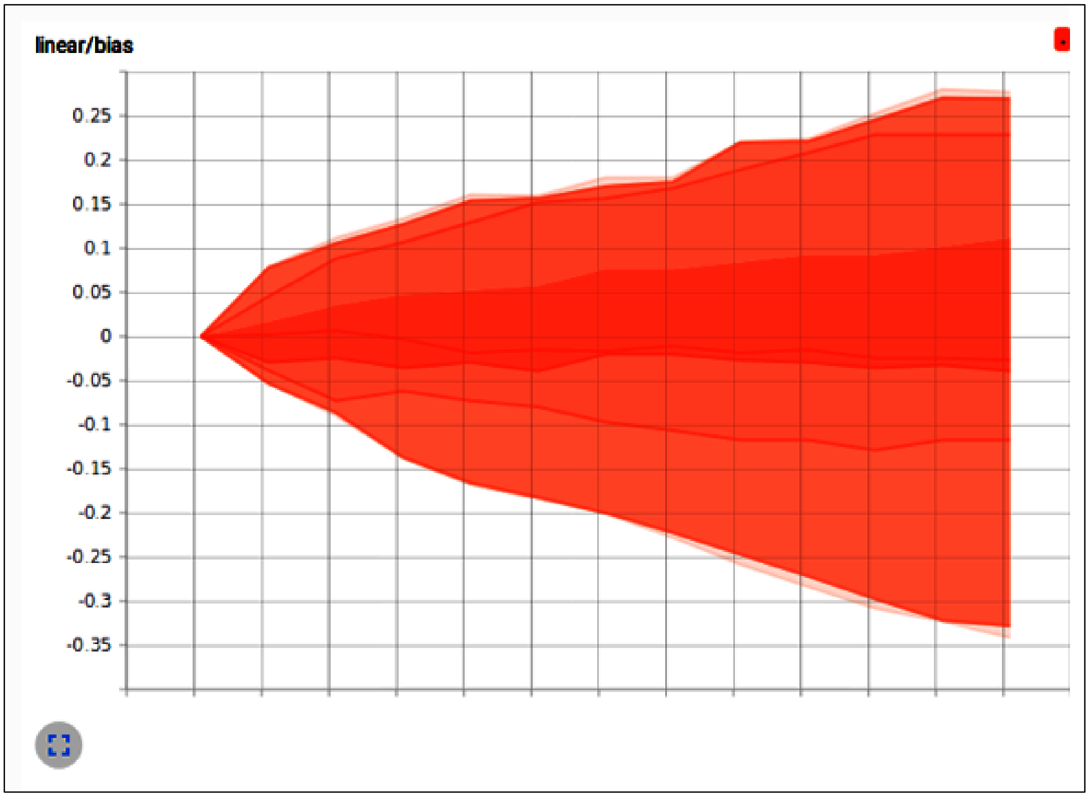

concat
Const
axis
?
concat_1
Const
axis
?
report_uninitial...
report_uninitial...
save
linear
init
12 tensors
12 tensors
126 tensors
scalar
scalar
global_step
init
scalar
2 tensors
MakeIter...
IteratorT...
IteratorG...
ModelDa...
IteratorV2
scalar
scalar
scalar
scalar
scalar
Optimize...
optimizatio...
scalar
RepeatD...
count
scalar
BatchDat...
batch_size
drop_remai...
scalar
ShuffleD...
buffer_size
seed
seed2
scalar
TensorSl...
component...
component...
component...
component...
component...
... 9 more
init
global_step
linear
group_deps
init_1
group_deps
group_de...
init
init_1
init_2
group_dep...
init_all_t...
group_dep...
init_3
group_dep...
group_de...
init_2
init_all_tables
init_3


Classification Task
Regression Task

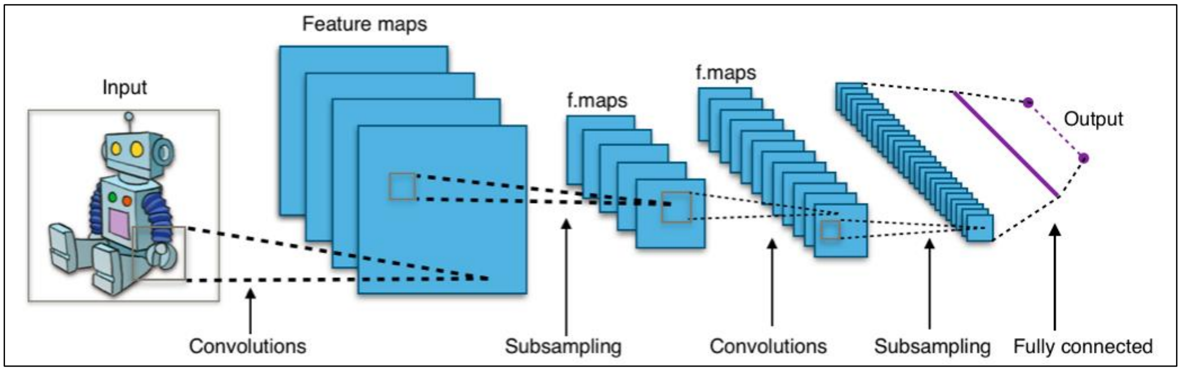

5
0
5
10
15
20
25
0
5
10
15
20
25

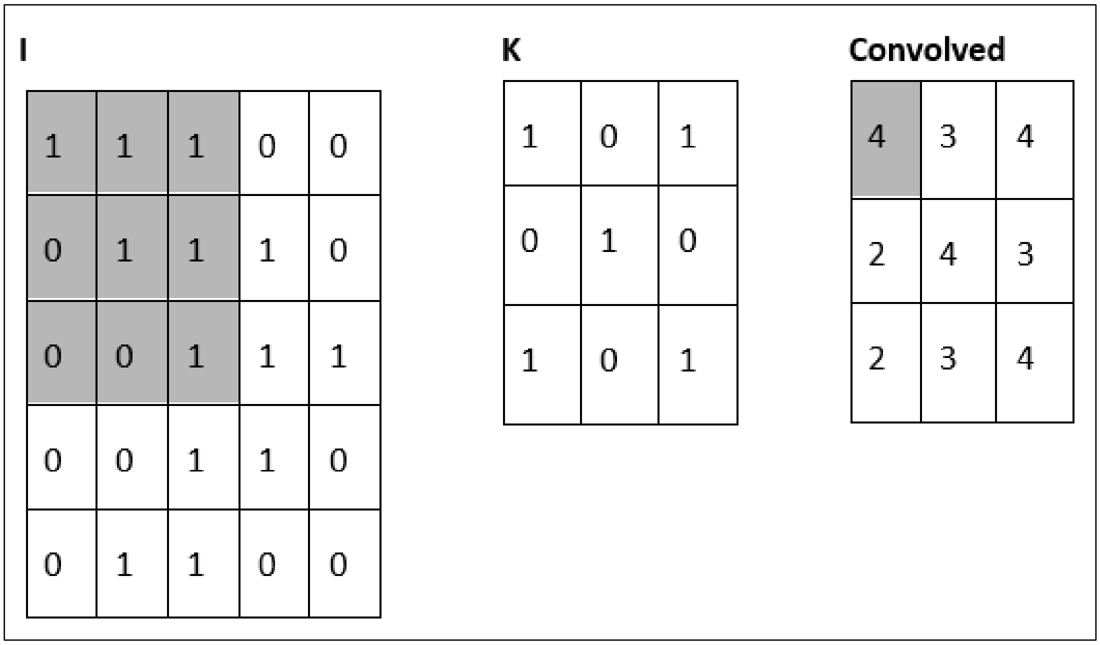

concat
Const
axis
concat_1
Const
axis
?
?
report_uninitial...
report_uninitial...
save
6 tensors
6 tensors
18 tensors
scalar
scalar
linear
init
scalar
2 tensors
2 tensors
global_step
init
random_...
n
scalar
enqueue_input
init
global_step
linear
group_deps
init_1
group_deps
group_de...
init
init_1
init_2
group_deps...
init_all_ta...
group_deps...
init_3
group_deps...
init_2
init_all_tabl...
init_3
group_de...


Loss
Accuracy
Average Loss
0.8
0.75
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0.3
Accuracy
0.91
0.9
0.89
0.88
0.87
0.86
0.85
0.84
0.83
0.82
0.81
0.8
0
20
40
60
80
100
120
140
Time Steps


linear/bias
0.25
0.2
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
-0.25
-0.3
-0.35

Chapter 4: Convolutional Neural Networks





I

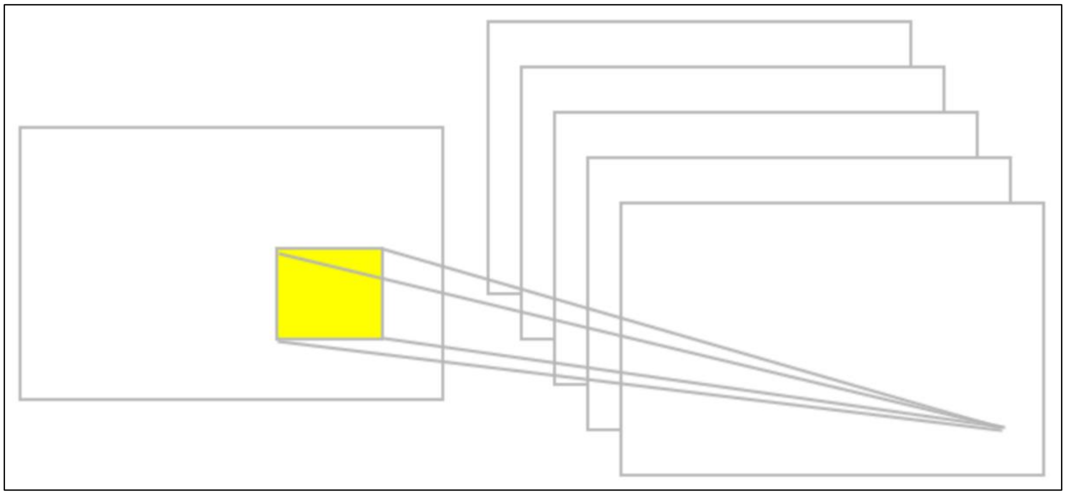
1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

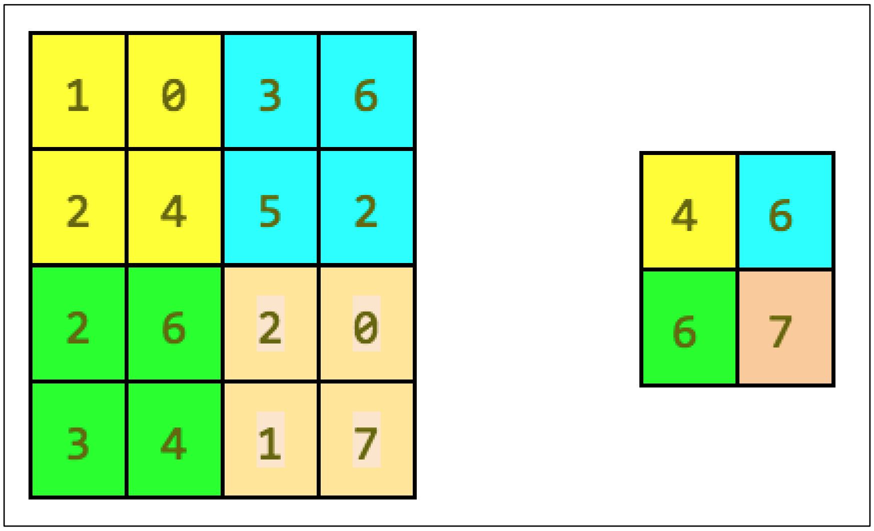
K

1	0	1
0	1	0
1	0	1

Convolved

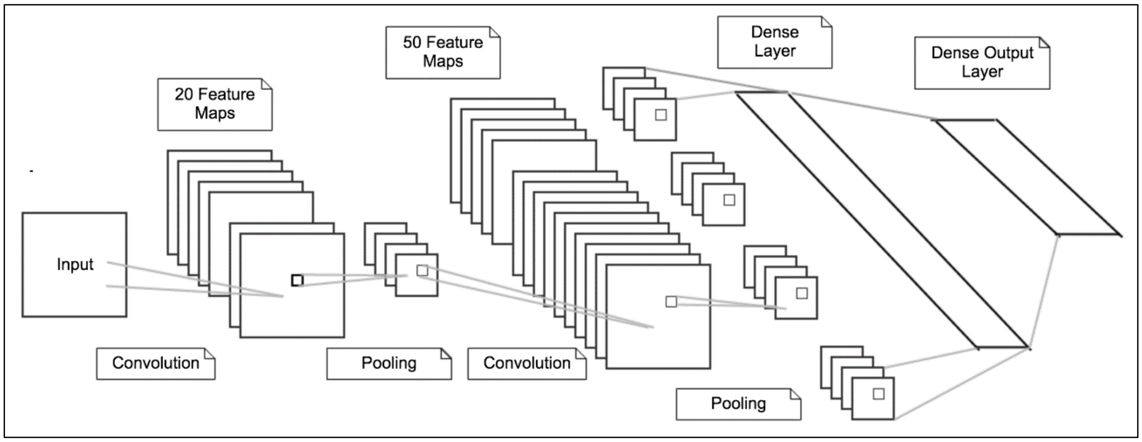
4	3	4
2	4	3
2	3	4

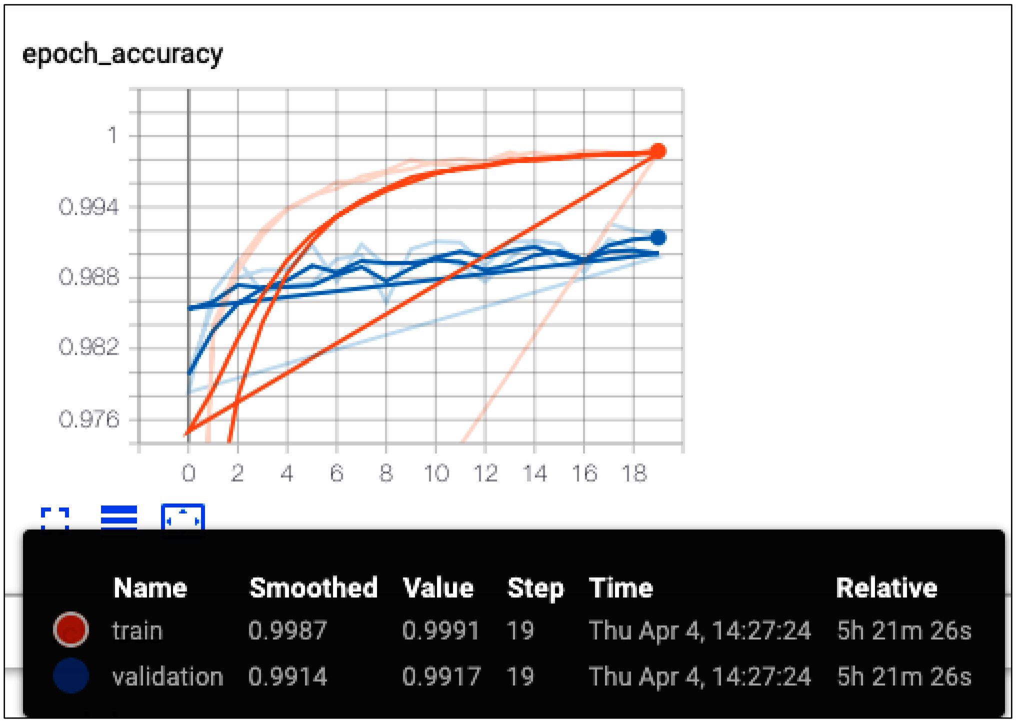




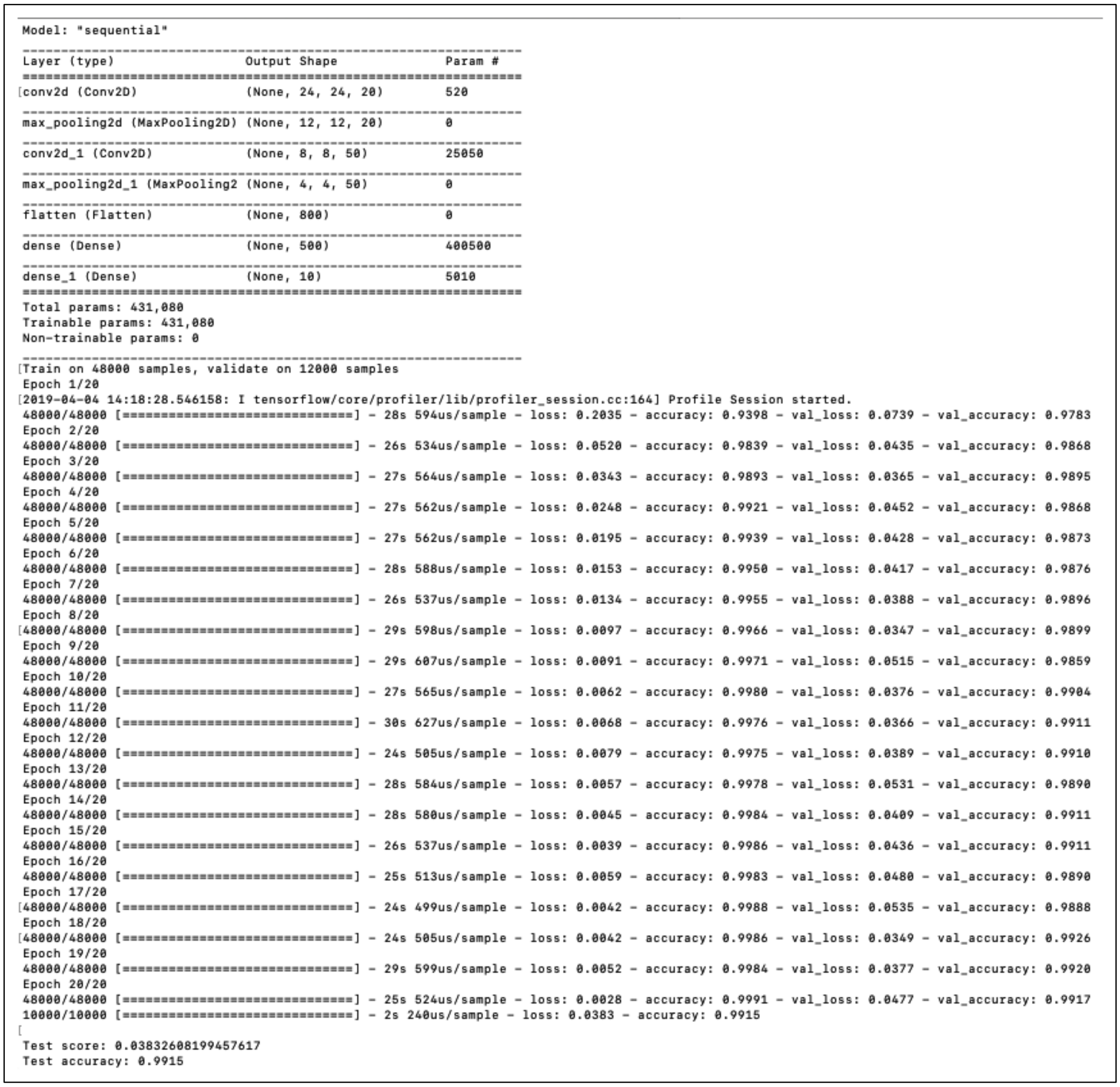
1	0	3	6
2	4	5	2
2	6	2	0
3	4	1	7

4	6
6	7

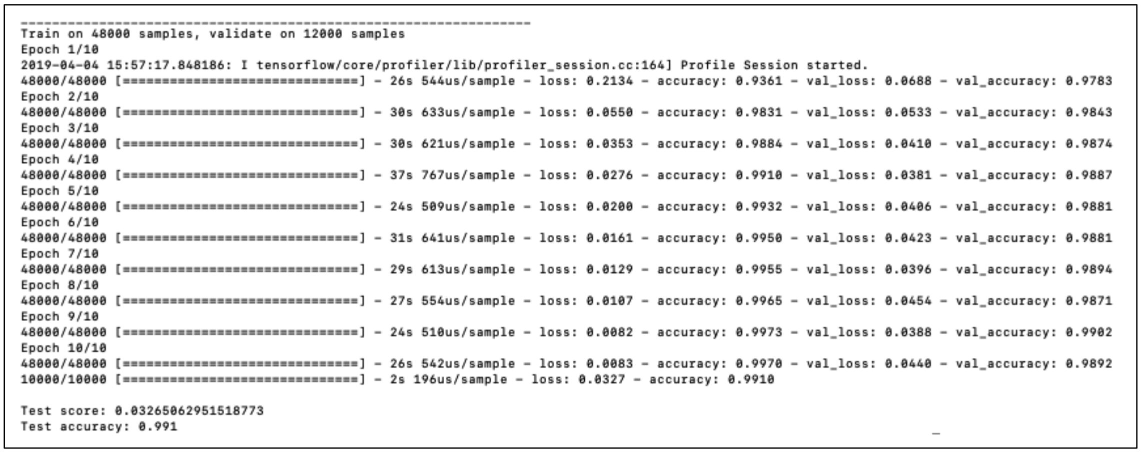




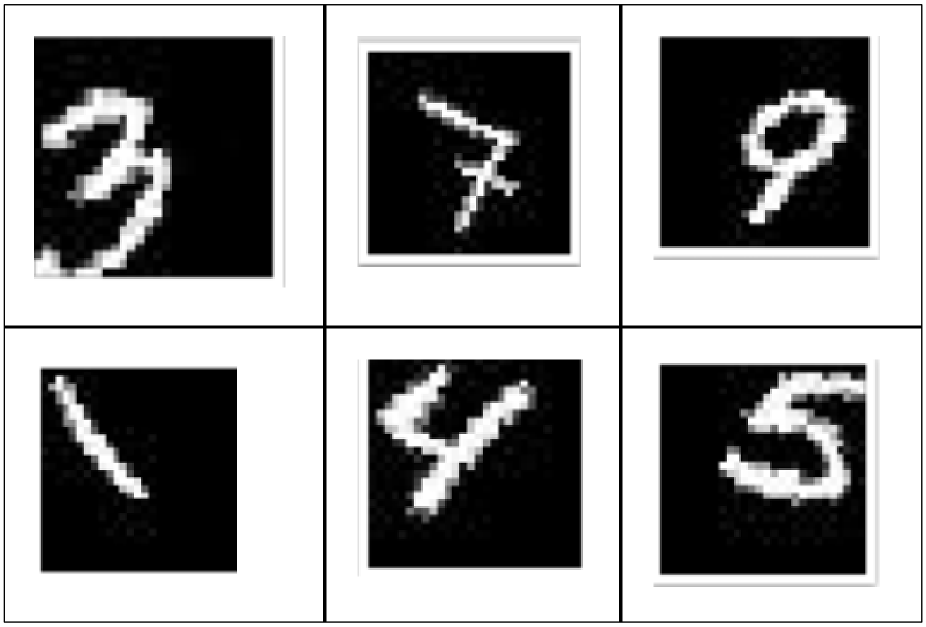
	Name	Smoothed	Value	Step	Time	Relative
●	train	0.9987	0.9991	19	Thu Apr 4, 14:27:24	5h 21m 26s
●	validation	0.9914	0.9917	19	Thu Apr 4, 14:27:24	5h 21m 26s

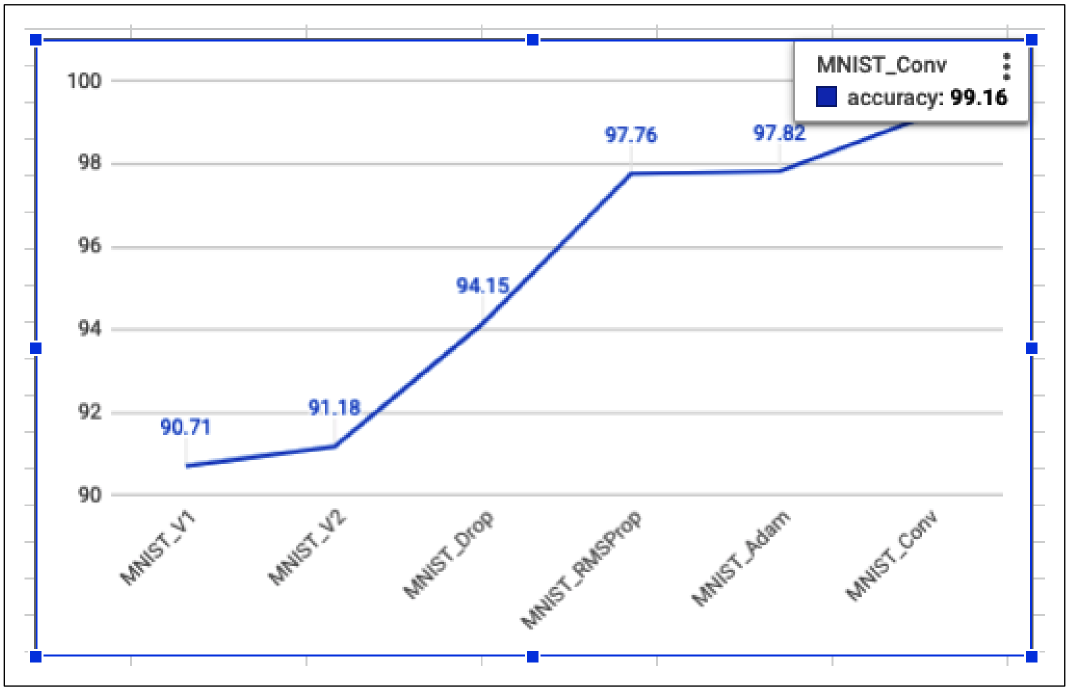


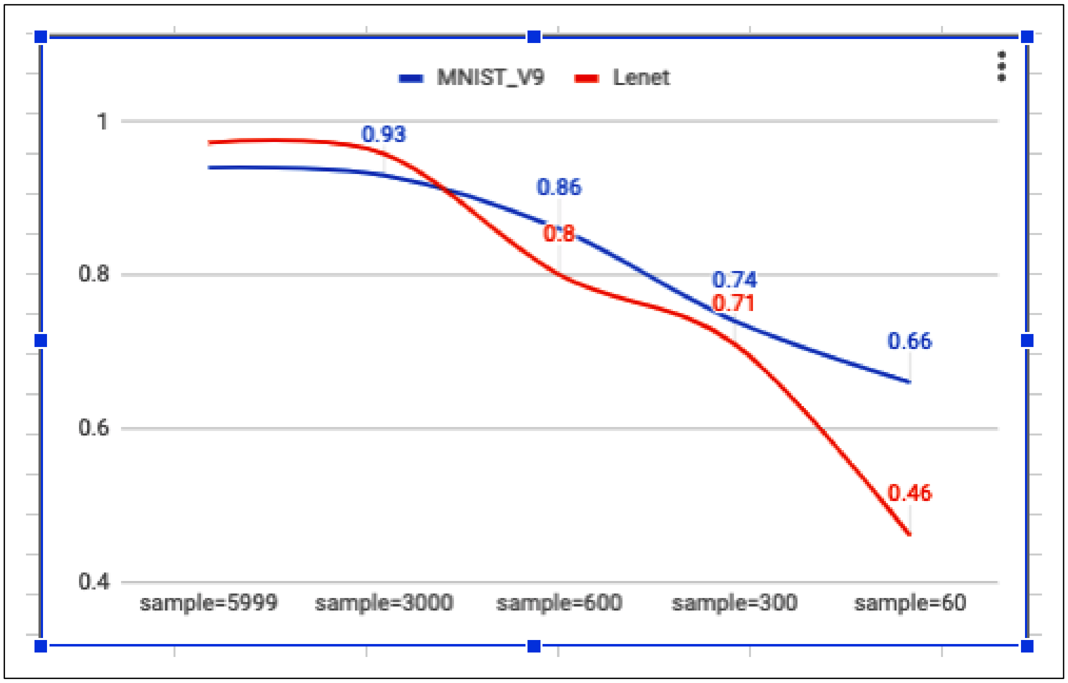
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
[conv2d (Conv2D)              (None, 24, 24, 20)        520
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 12, 12, 20)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 8, 8, 50)          25050
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 4, 4, 50)          0
_________________________________________________________________
flatten (Flatten)            (None, 800)               0
_________________________________________________________________
dense (Dense)                (None, 500)               400500
_________________________________________________________________
dense_1 (Dense)              (None, 10)                5010
=================================================================
Total params: 431,080
Trainable params: 431,080
Non-trainable params: 0
_________________________________________________________________
[Train on 48000 samples, validate on 12000 samples
Epoch 1/20
[2019-04-04 14:18:28.546158: I tensorflow/core/profiler/lib/profiler_session.cc:164] Profile Session started.
48000/48000 [==============================] - 28s 594us/sample - loss: 0.2035 - accuracy: 0.9398 - val_loss: 0.0739 - val_accuracy: 0.9783
Epoch 2/20
48000/48000 [==============================] - 26s 534us/sample - loss: 0.0520 - accuracy: 0.9839 - val_loss: 0.0435 - val_accuracy: 0.9868
Epoch 3/20
48000/48000 [==============================] - 27s 564us/sample - loss: 0.0343 - accuracy: 0.9893 - val_loss: 0.0365 - val_accuracy: 0.9895
Epoch 4/20
48000/48000 [==============================] - 27s 562us/sample - loss: 0.0248 - accuracy: 0.9921 - val_loss: 0.0452 - val_accuracy: 0.9868
Epoch 5/20
48000/48000 [==============================] - 27s 562us/sample - loss: 0.0195 - accuracy: 0.9939 - val_loss: 0.0428 - val_accuracy: 0.9873
Epoch 6/20
48000/48000 [==============================] - 28s 588us/sample - loss: 0.0153 - accuracy: 0.9950 - val_loss: 0.0417 - val_accuracy: 0.9876
Epoch 7/20
48000/48000 [==============================] - 26s 537us/sample - loss: 0.0134 - accuracy: 0.9955 - val_loss: 0.0388 - val_accuracy: 0.9896
Epoch 8/20
[48000/48000 [==============================] - 29s 598us/sample - loss: 0.0097 - accuracy: 0.9966 - val_loss: 0.0347 - val_accuracy: 0.9899
Epoch 9/20
48000/48000 [==============================] - 29s 607us/sample - loss: 0.0091 - accuracy: 0.9971 - val_loss: 0.0515 - val_accuracy: 0.9859
Epoch 10/20
48000/48000 [==============================] - 27s 565us/sample - loss: 0.0062 - accuracy: 0.9980 - val_loss: 0.0376 - val_accuracy: 0.9904
Epoch 11/20
48000/48000 [==============================] - 30s 627us/sample - loss: 0.0068 - accuracy: 0.9976 - val_loss: 0.0366 - val_accuracy: 0.9911
Epoch 12/20
48000/48000 [==============================] - 24s 505us/sample - loss: 0.0079 - accuracy: 0.9975 - val_loss: 0.0389 - val_accuracy: 0.9910
Epoch 13/20
48000/48000 [==============================] - 28s 584us/sample - loss: 0.0057 - accuracy: 0.9978 - val_loss: 0.0531 - val_accuracy: 0.9890
Epoch 14/20
48000/48000 [==============================] - 28s 580us/sample - loss: 0.0045 - accuracy: 0.9984 - val_loss: 0.0409 - val_accuracy: 0.9911
Epoch 15/20
48000/48000 [==============================] - 26s 537us/sample - loss: 0.0039 - accuracy: 0.9986 - val_loss: 0.0436 - val_accuracy: 0.9911
Epoch 16/20
48000/48000 [==============================] - 25s 513us/sample - loss: 0.0059 - accuracy: 0.9983 - val_loss: 0.0480 - val_accuracy: 0.9890
Epoch 17/20
[48000/48000 [==============================] - 24s 499us/sample - loss: 0.0042 - accuracy: 0.9988 - val_loss: 0.0535 - val_accuracy: 0.9888
Epoch 18/20
[48000/48000 [==============================] - 24s 505us/sample - loss: 0.0042 - accuracy: 0.9986 - val_loss: 0.0349 - val_accuracy: 0.9926
Epoch 19/20
48000/48000 [==============================] - 29s 599us/sample - loss: 0.0052 - accuracy: 0.9984 - val_loss: 0.0377 - val_accuracy: 0.9920
Epoch 20/20
48000/48000 [==============================] - 25s 524us/sample - loss: 0.0028 - accuracy: 0.9991 - val_loss: 0.0477 - val_accuracy: 0.9917
10000/10000 [==============================] - 2s 240us/sample - loss: 0.0383 - accuracy: 0.9915
[
Test score: 0.03832608199457617
Test accuracy: 0.9915
```

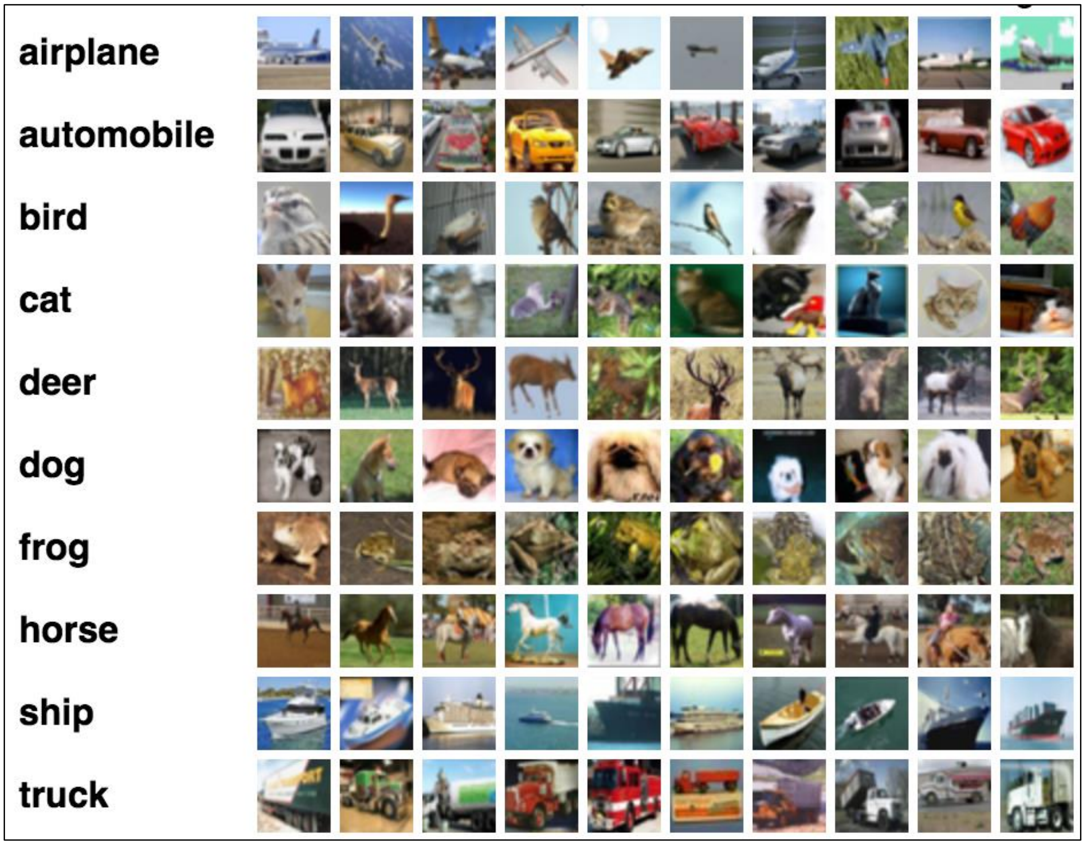


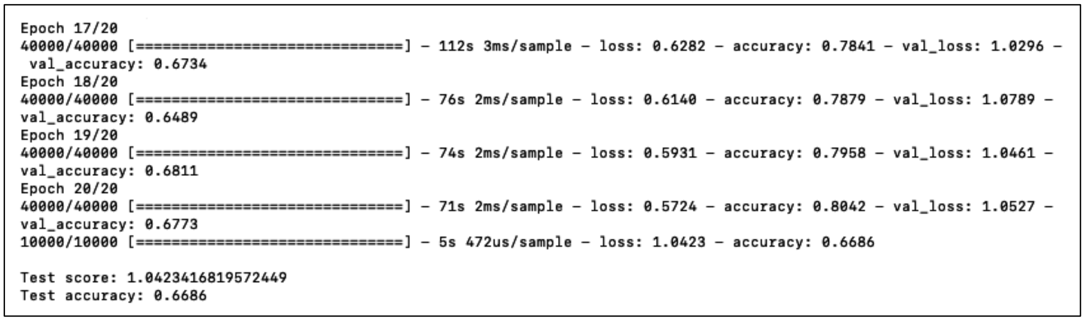
```
-----------------------------------------------------------------
Train on 48000 samples, validate on 12000 samples
Epoch 1/10
2019-04-04 15:57:17.848186: I tensorflow/core/profiler/lib/profiler_session.cc:164] Profile Session started.
48000/48000 [==============================] - 26s 544us/sample - loss: 0.2134 - accuracy: 0.9361 - val_loss: 0.0688 - val_accuracy: 0.9783
Epoch 2/10
48000/48000 [==============================] - 30s 633us/sample - loss: 0.0550 - accuracy: 0.9831 - val_loss: 0.0533 - val_accuracy: 0.9843
Epoch 3/10
48000/48000 [==============================] - 30s 621us/sample - loss: 0.0353 - accuracy: 0.9884 - val_loss: 0.0410 - val_accuracy: 0.9874
Epoch 4/10
48000/48000 [==============================] - 37s 767us/sample - loss: 0.0276 - accuracy: 0.9910 - val_loss: 0.0381 - val_accuracy: 0.9887
Epoch 5/10
48000/48000 [==============================] - 24s 509us/sample - loss: 0.0200 - accuracy: 0.9932 - val_loss: 0.0406 - val_accuracy: 0.9881
Epoch 6/10
48000/48000 [==============================] - 31s 641us/sample - loss: 0.0161 - accuracy: 0.9950 - val_loss: 0.0423 - val_accuracy: 0.9881
Epoch 7/10
48000/48000 [==============================] - 29s 613us/sample - loss: 0.0129 - accuracy: 0.9955 - val_loss: 0.0396 - val_accuracy: 0.9894
Epoch 8/10
48000/48000 [==============================] - 27s 554us/sample - loss: 0.0107 - accuracy: 0.9965 - val_loss: 0.0454 - val_accuracy: 0.9871
Epoch 9/10
48000/48000 [==============================] - 24s 510us/sample - loss: 0.0082 - accuracy: 0.9973 - val_loss: 0.0388 - val_accuracy: 0.9902
Epoch 10/10
48000/48000 [==============================] - 26s 542us/sample - loss: 0.0083 - accuracy: 0.9970 - val_loss: 0.0440 - val_accuracy: 0.9892
10000/10000 [==============================] - 2s 196us/sample - loss: 0.0327 - accuracy: 0.9910

Test score: 0.03265062951518773
Test accuracy: 0.991
```



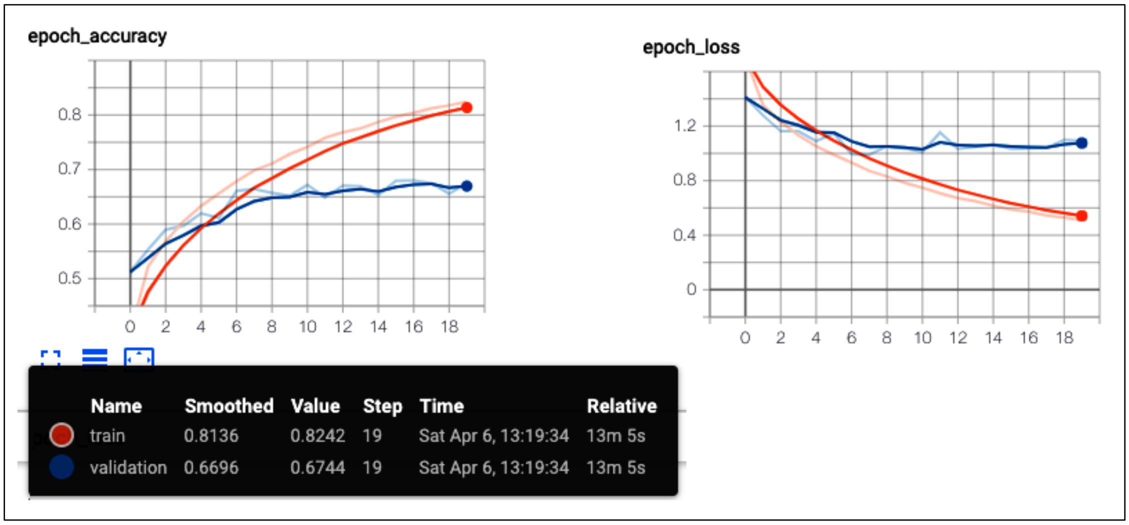

100
98
96
94
92
90
90.71
91.18
94.15
97.76
97.82
MNIST_Conv
accuracy: 99.16
MNIST_V1
MNIST_V2
MNIST_Drop
MNIST_RMSProp
MNIST_Adam
MNIST_Conv


MNIST_V9
Lenet
1
0.8
0.6
0.4
0.93
0.86
0.8
0.74
0.71
0.66
0.46
sample=5999
sample=3000
sample=600
sample=300
sample=60

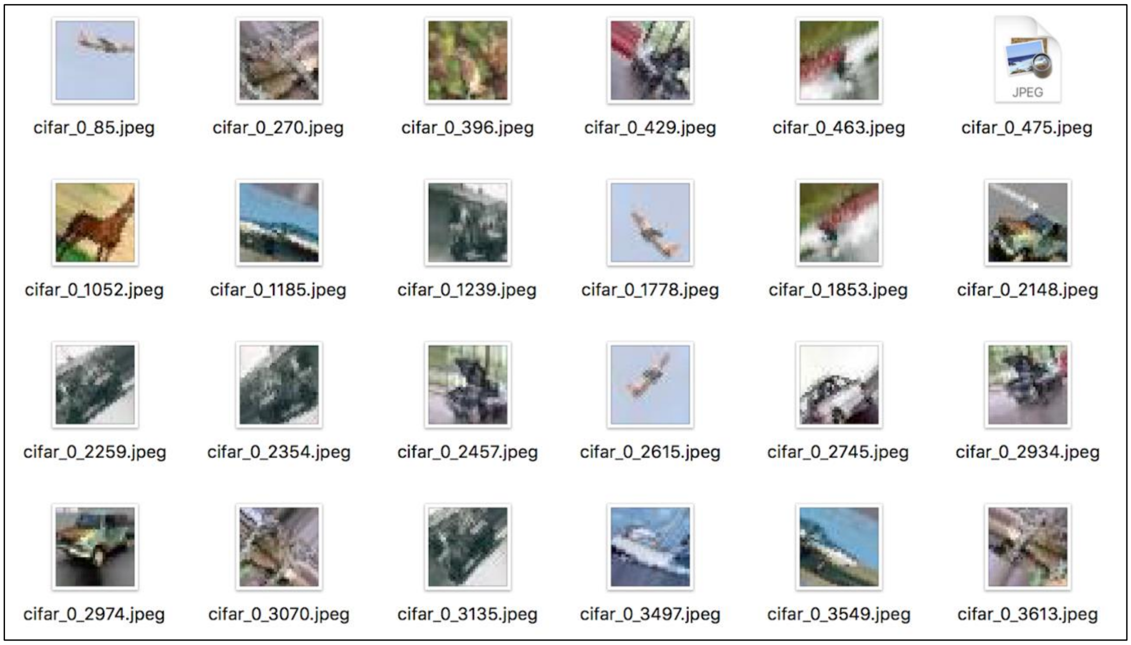


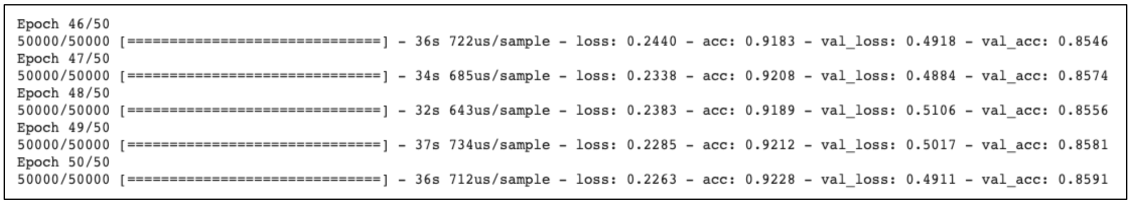


```
Epoch 17/20
40000/40000 [==============================] - 112s 3ms/sample - loss: 0.6282 - accuracy: 0.7841 - val_loss: 1.0296 -
 val_accuracy: 0.6734
Epoch 18/20
40000/40000 [==============================] - 76s 2ms/sample - loss: 0.6140 - accuracy: 0.7879 - val_loss: 1.0789 -
val_accuracy: 0.6489
Epoch 19/20
40000/40000 [==============================] - 74s 2ms/sample - loss: 0.5931 - accuracy: 0.7958 - val_loss: 1.0461 -
val_accuracy: 0.6811
Epoch 20/20
40000/40000 [==============================] - 71s 2ms/sample - loss: 0.5724 - accuracy: 0.8042 - val_loss: 1.0527 -
val_accuracy: 0.6773
10000/10000 [==============================] - 5s 472us/sample - loss: 1.0423 - accuracy: 0.6686

Test score: 1.0423416819572449
Test accuracy: 0.6686
```

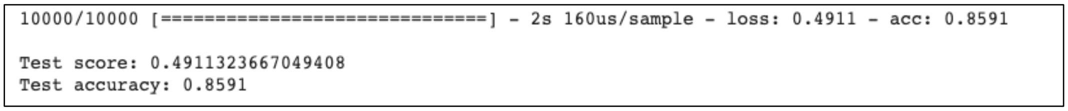


	Name	Smoothed	Value	Step	Time	Relative
●	train	0.8136	0.8242	19	Sat Apr 6, 13:19:34	13m 5s
●	validation	0.6696	0.6744	19	Sat Apr 6, 13:19:34	13m 5s

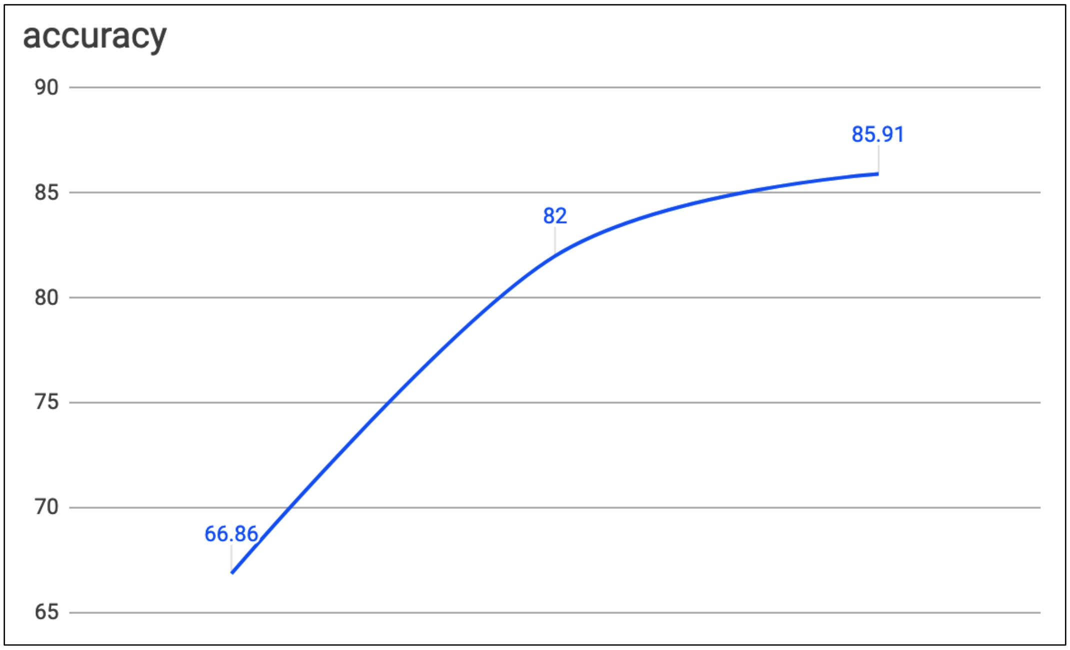


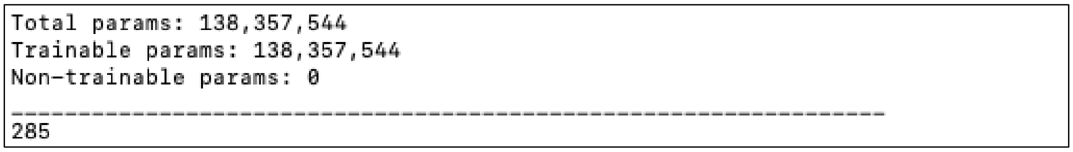


```
Epoch 46/50
50000/50000 [==============================] - 36s 722us/sample - loss: 0.2440 - acc: 0.9183 - val_loss: 0.4918 - val_acc: 0.8546
Epoch 47/50
50000/50000 [==============================] - 34s 685us/sample - loss: 0.2338 - acc: 0.9208 - val_loss: 0.4884 - val_acc: 0.8574
Epoch 48/50
50000/50000 [==============================] - 32s 643us/sample - loss: 0.2383 - acc: 0.9189 - val_loss: 0.5106 - val_acc: 0.8556
Epoch 49/50
50000/50000 [==============================] - 37s 734us/sample - loss: 0.2285 - acc: 0.9212 - val_loss: 0.5017 - val_acc: 0.8581
Epoch 50/50
50000/50000 [==============================] - 36s 712us/sample - loss: 0.2263 - acc: 0.9228 - val_loss: 0.4911 - val_acc: 0.8591
```

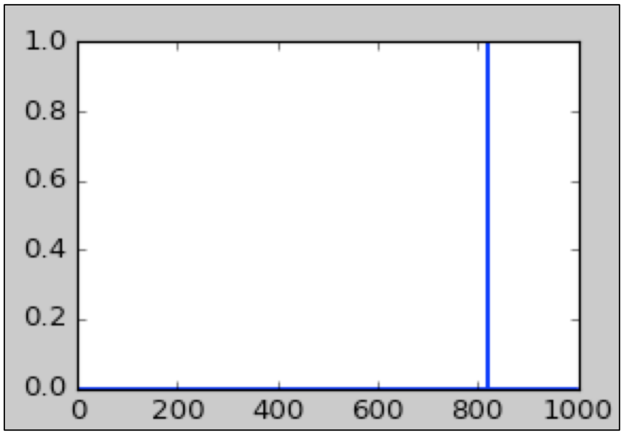


```
10000/10000 [==============================] - 2s 160us/sample - loss: 0.4911 - acc: 0.8591

Test score: 0.4911323667049408
Test accuracy: 0.8591
```



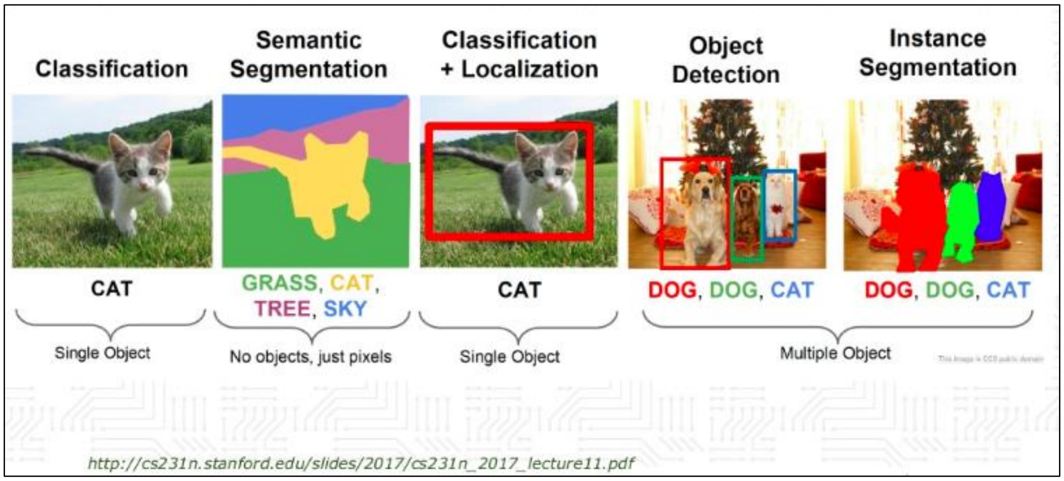


```
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
_________________________________________________________________
285
```

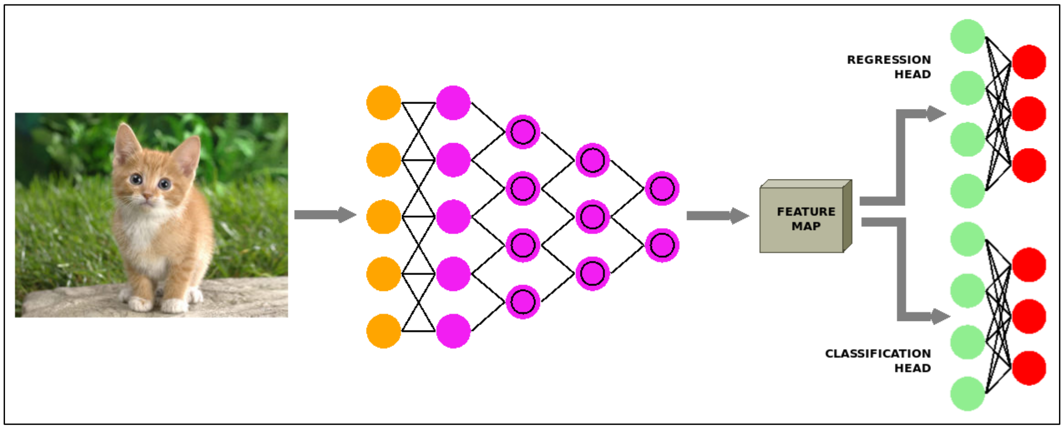


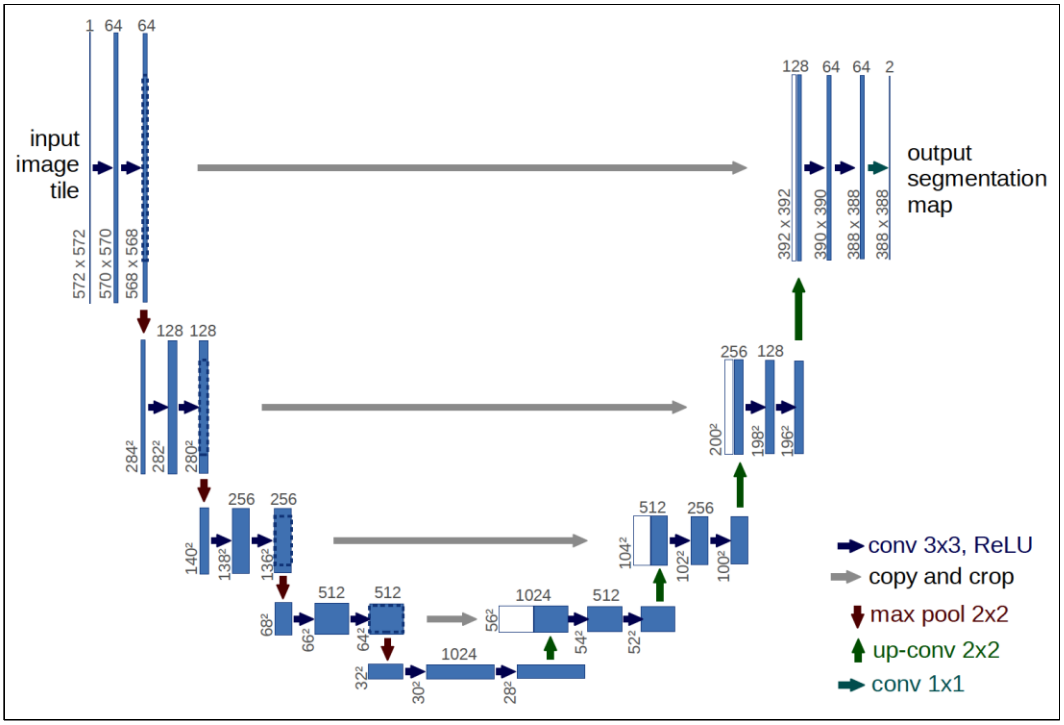
78 468

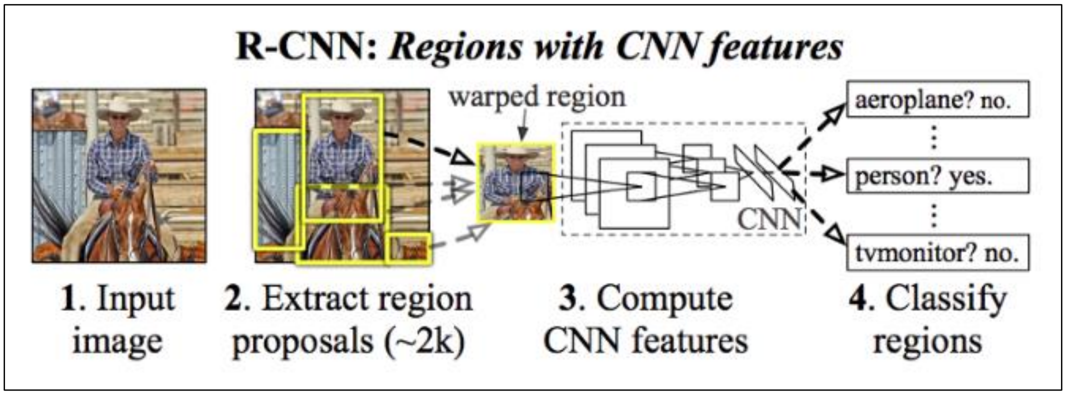
Chapter 5: Advanced Convolutional Neural Networks

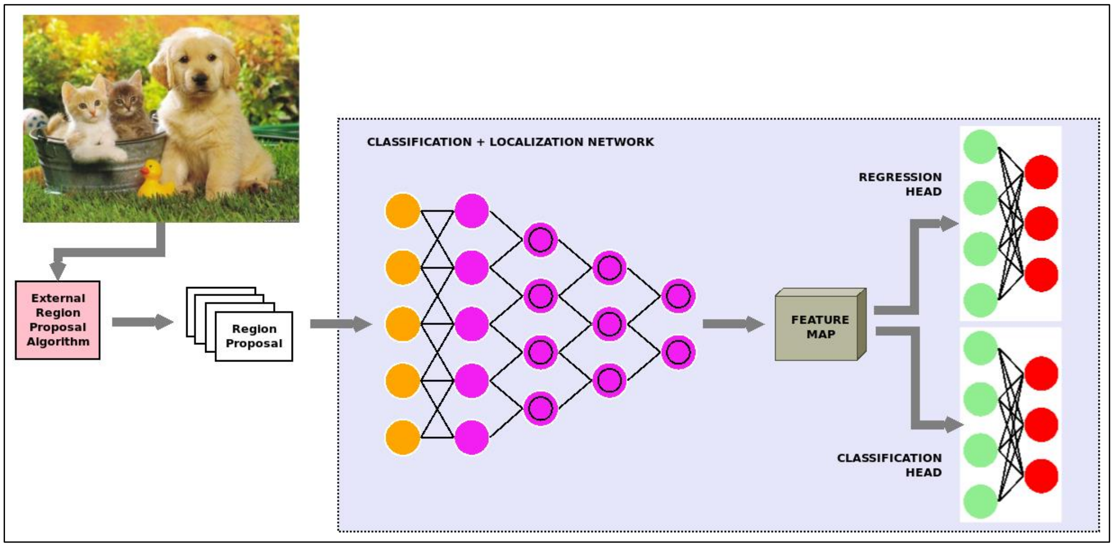


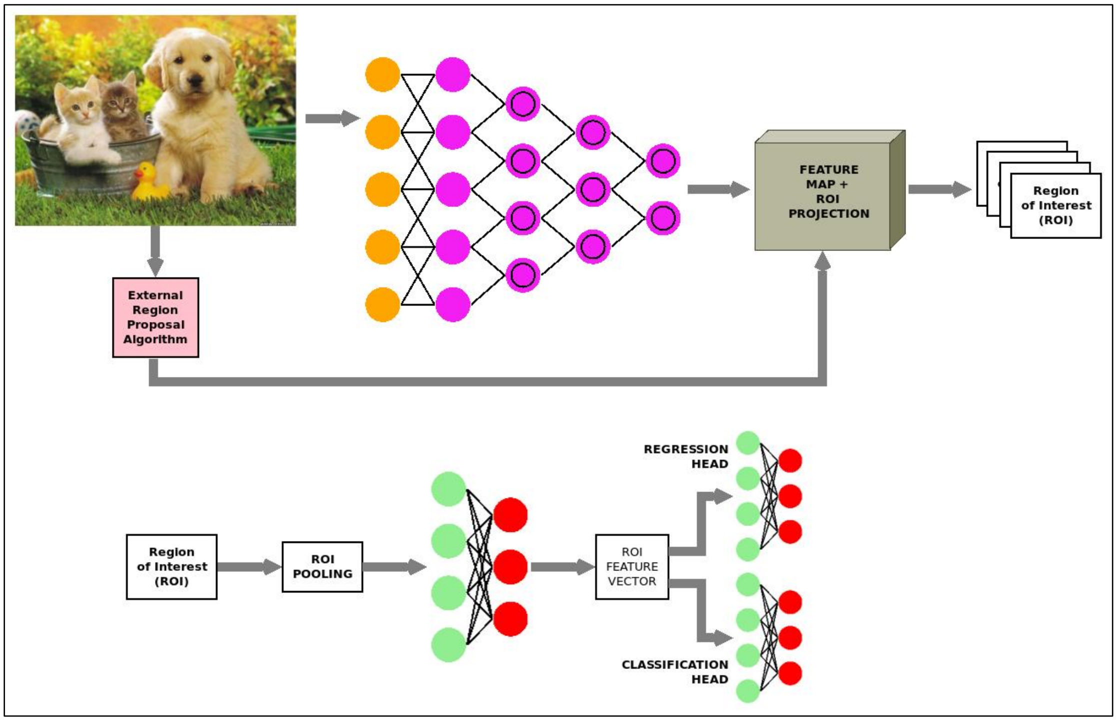
http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

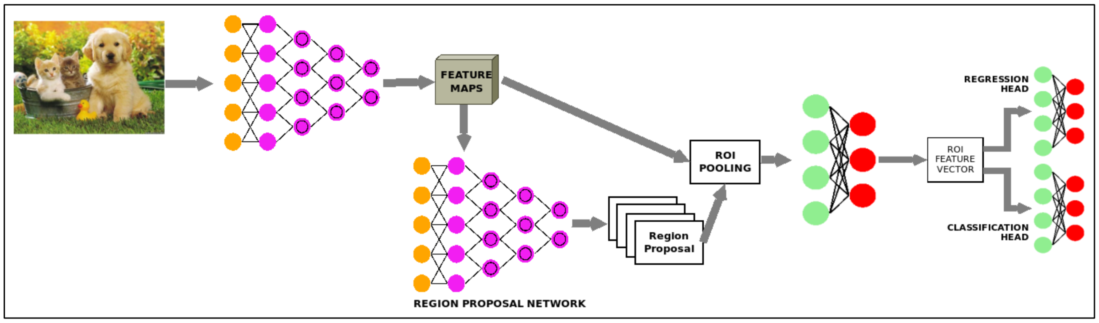


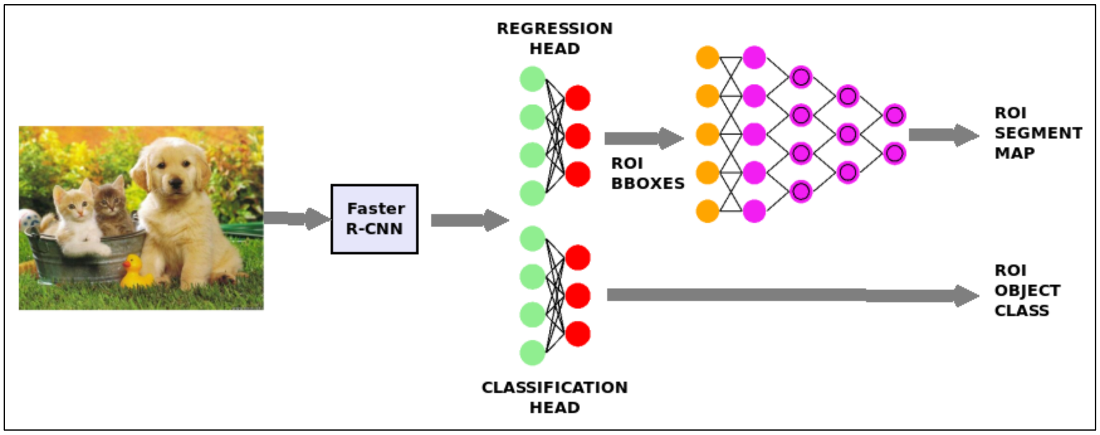

1
64
64
input
image
tile
572 x 572
570 x 570
568 x 568
128
128
284^2
282^2
280^2
256
256
140^2
138^2
136^2
512
512
68^2
66^2
64^2
1024
32^2
30^2
28^2
1024
512
56^2
54^2
52^2
512
256
104^2
102^2
100^2
256
128
200^2
198^2
196^2
128
64
64
2
392 x 392
390 x 390
388 x 388
388 x 388
output
segmentation
map
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1

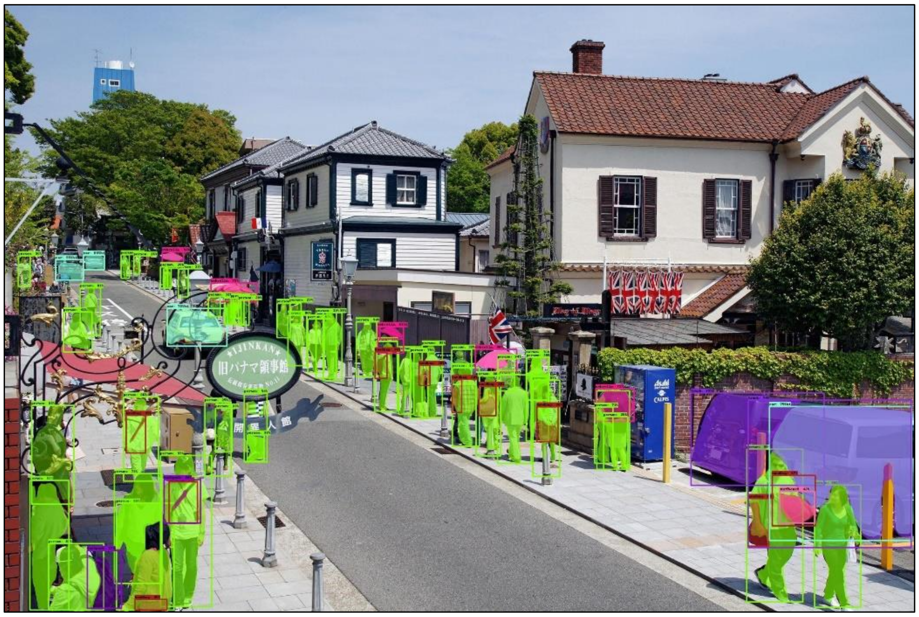

R-CNN: *Regions with CNN features*
warped region
CNN
aeroplane? no.
person? yes.
tvmonitor? no.
1. Input
image
2. Extract region
proposals (~2k)
3. Compute
CNN features
4. Classify
regions

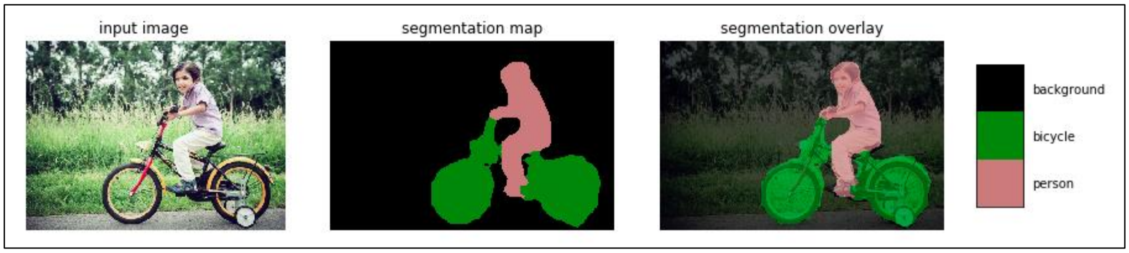

External Region Proposal Algorithm
Region Proposal
CLASSIFICATION + LOCALIZATION NETWORK
FEATURE MAP
REGRESSION HEAD
CLASSIFICATION HEAD

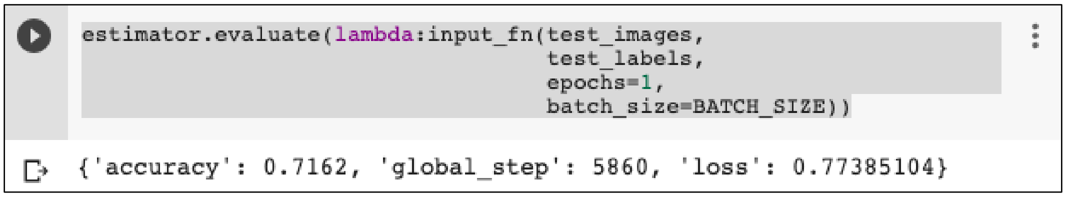

External Region Proposal Algorithm
FEATURE MAP + ROI PROJECTION
Region of Interest (ROI)
Region of Interest (ROI)
ROI POOLING
ROI FEATURE VECTOR
REGRESSION HEAD
CLASSIFICATION HEAD


FEATURE MAPS
ROI POOLING
Region Proposal
REGION PROPOSAL NETWORK
ROI FEATURE VECTOR
REGRESSION HEAD
CLASSIFICATION HEAD


Faster R-CNN
REGRESSION HEAD
ROI BBOXES
ROI SEGMENT MAP
ROI OBJECT CLASS
CLASSIFICATION HEAD

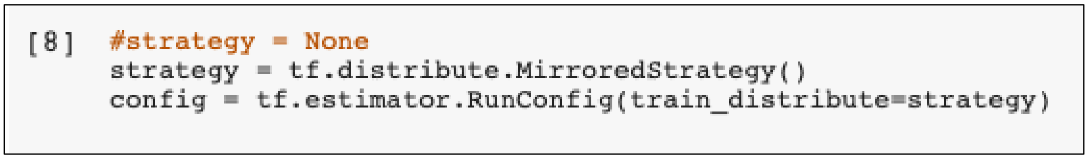

IJINKAN
旧パナマ領事館



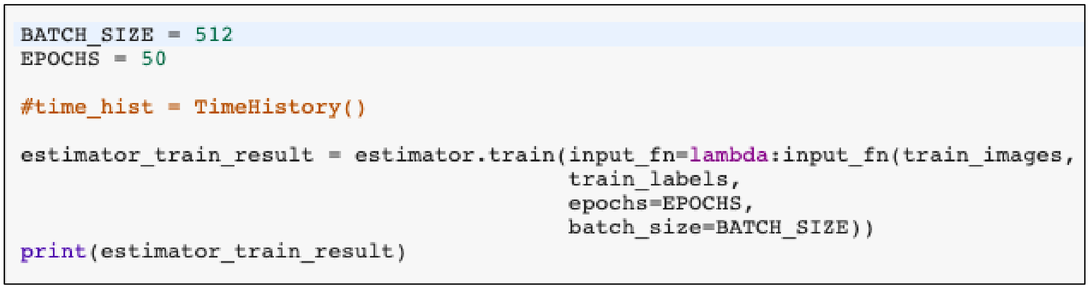


```
estimator.evaluate(lambda:input_fn(test_images,
                                   test_labels,
                                   epochs=1,
                                   batch_size=BATCH_SIZE))
```

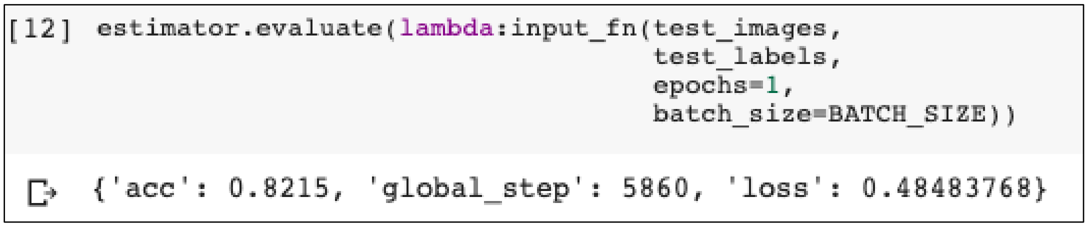
```
{'accuracy': 0.7162, 'global_step': 5860, 'loss': 0.77385104}
```



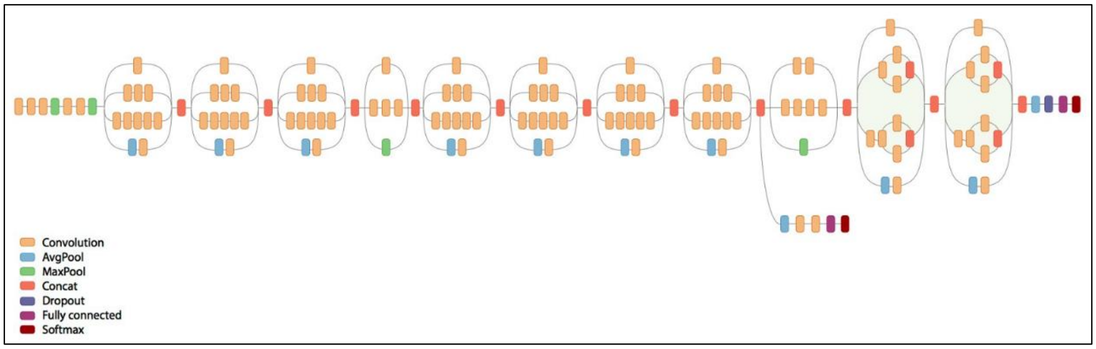
```
[8]  #strategy = None
     strategy = tf.distribute.MirroredStrategy()
     config = tf.estimator.RunConfig(train_distribute=strategy)
```

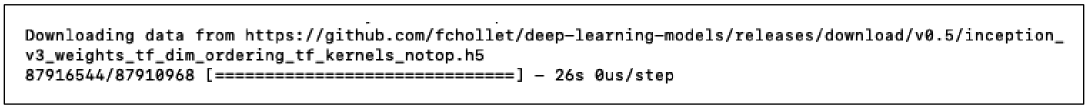


```
BATCH_SIZE = 512
EPOCHS = 50

#time_hist = TimeHistory()

estimator_train_result = estimator.train(input_fn=lambda:input_fn(train_images,
                                         train_labels,
                                         epochs=EPOCHS,
                                         batch_size=BATCH_SIZE))
print(estimator_train_result)
```

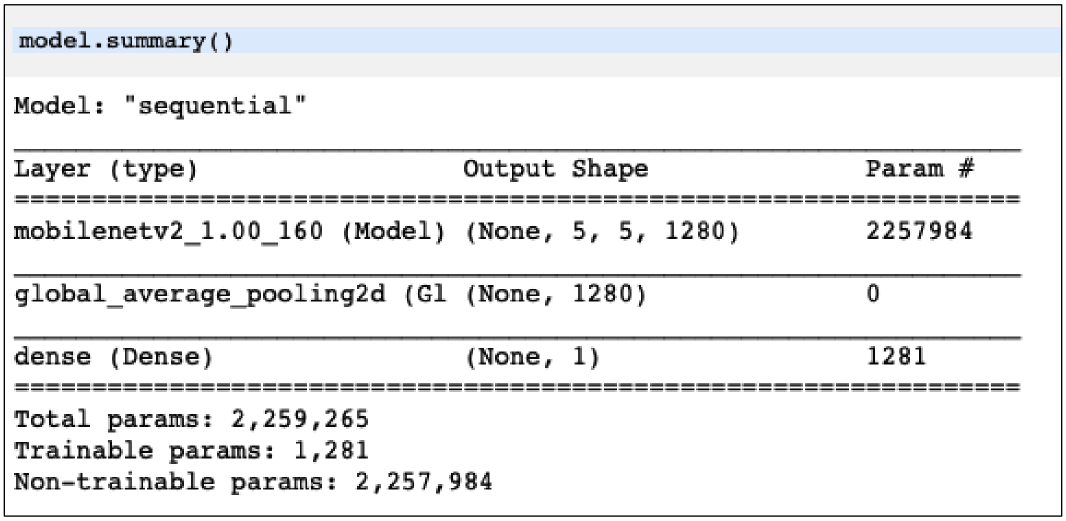


```
[12] estimator.evaluate(lambda:input_fn(test_images,
                                        test_labels,
                                        epochs=1,
                                        batch_size=BATCH_SIZE))

{'acc': 0.8215, 'global_step': 5860, 'loss': 0.48483768}
```



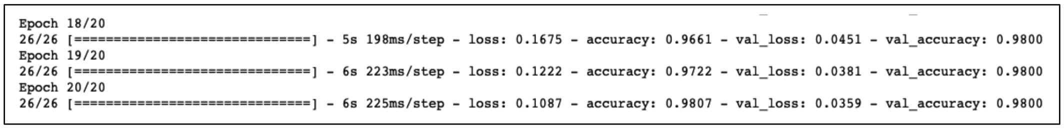


```
Downloading data from https://github.com/fchollet/deep-learning-models/releases/download/v0.5/inception_
v3_weights_tf_dim_ordering_tf_kernels_notop.h5
87916544/87910968 [==============================] - 26s 0us/step
```

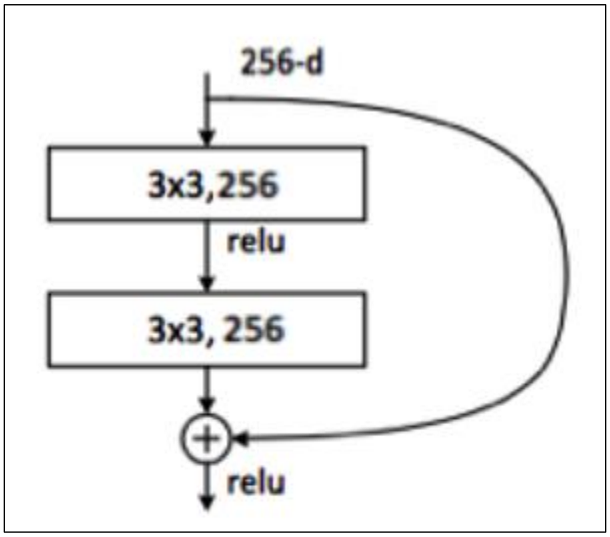


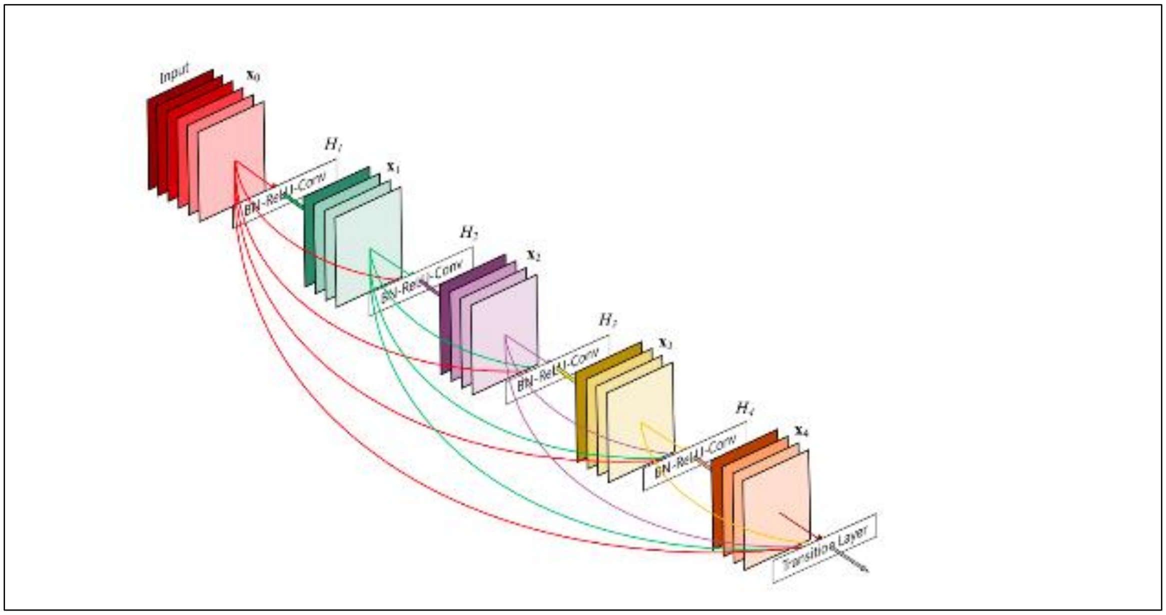
```
model.summary()
```

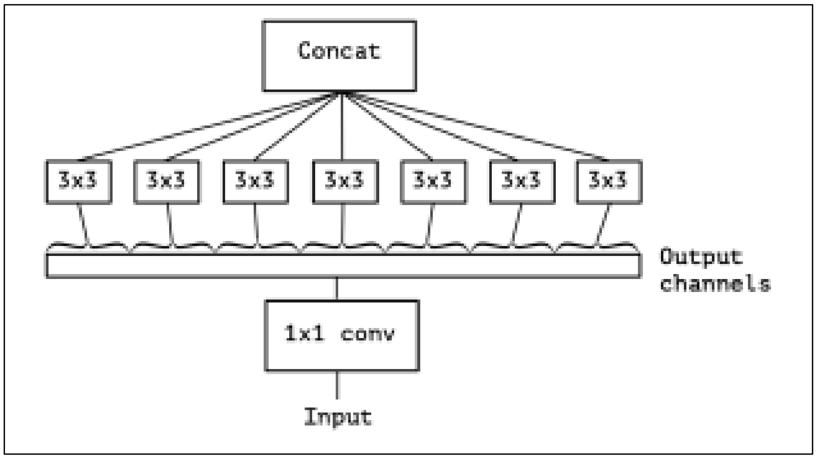
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
mobilenetv2_1.00_160 (Model) (None, 5, 5, 1280)        2257984
_________________________________________________________________
global_average_pooling2d (Gl (None, 1280)              0
_________________________________________________________________
dense (Dense)                (None, 1)                 1281
=================================================================
Total params: 2,259,265
Trainable params: 1,281
Non-trainable params: 2,257,984
```

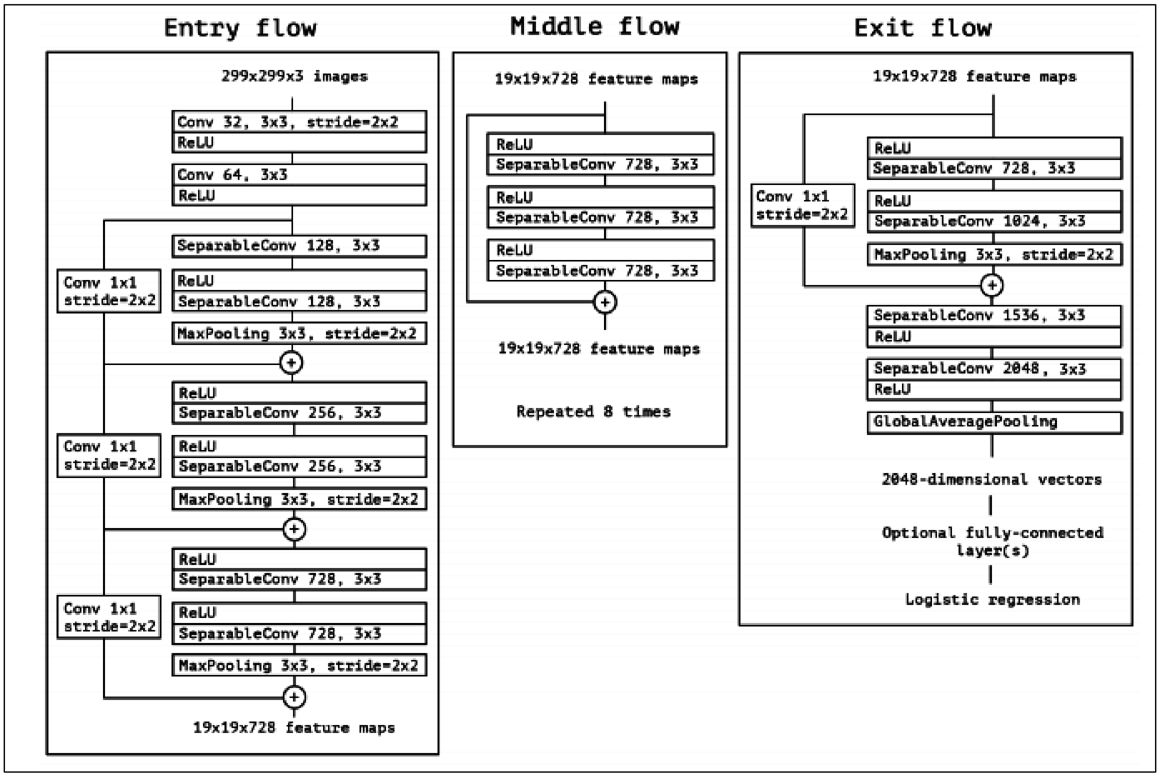


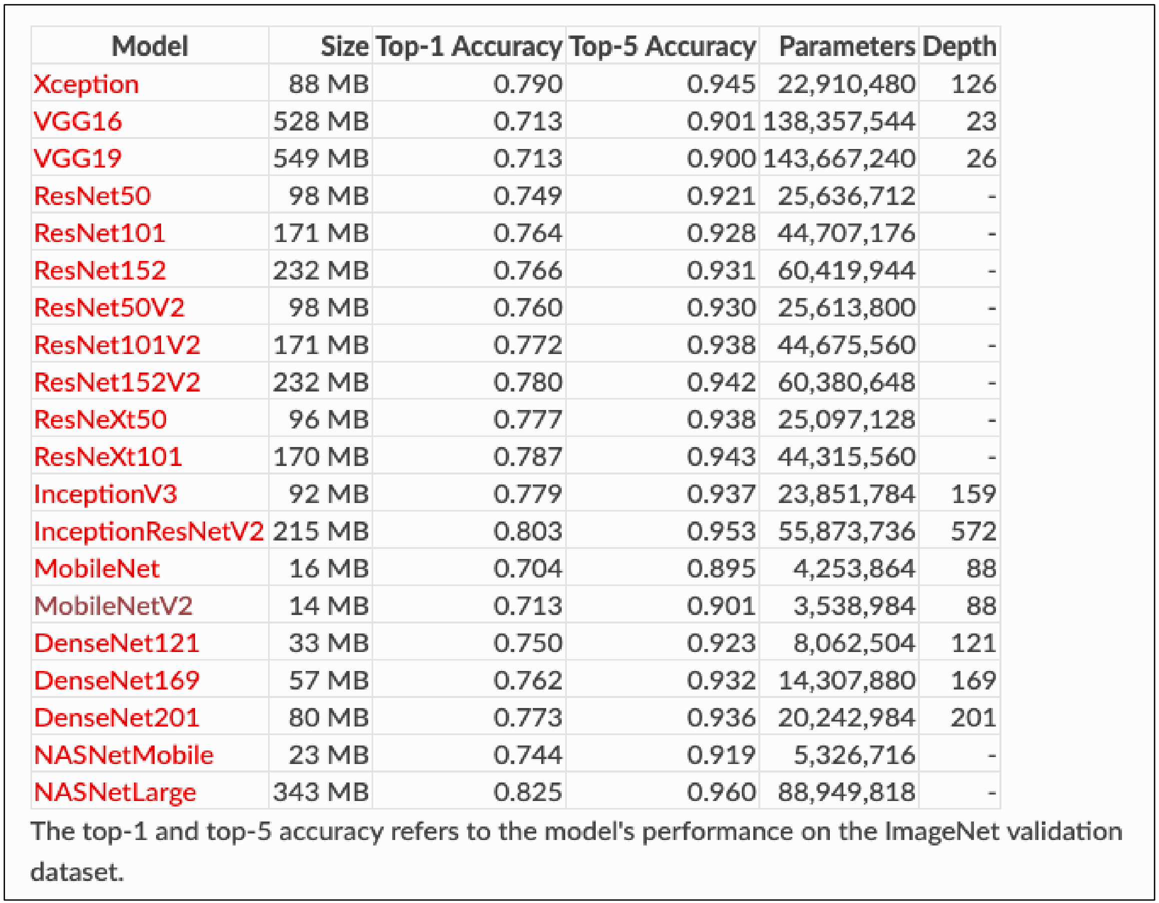
```
Epoch 18/20
26/26 [==============================] - 5s 198ms/step - loss: 0.1675 - accuracy: 0.9661 - val_loss: 0.0451 - val_accuracy: 0.9800
Epoch 19/20
26/26 [==============================] - 6s 223ms/step - loss: 0.1222 - accuracy: 0.9722 - val_loss: 0.0381 - val_accuracy: 0.9800
Epoch 20/20
26/26 [==============================] - 6s 225ms/step - loss: 0.1087 - accuracy: 0.9807 - val_loss: 0.0359 - val_accuracy: 0.9800
```




Input
$\mathbf{x}_0$
H_1
BN-ReLU-Conv
$\mathbf{x}_1$
H_2
BN-ReLU-Conv
$\mathbf{x}_2$
H_3
BN-ReLU-Conv
$\mathbf{x}_3$
H_4
BN-ReLU-Conv
$\mathbf{x}_4$
Transition Layer

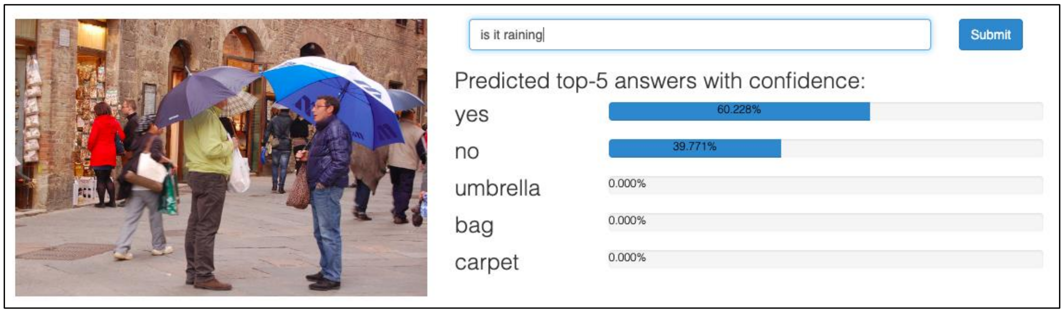

Concat
3x3
3x3
3x3
3x3
3x3
3x3
3x3
Output channels
1x1 conv
Input


Entry flow
299x299x3 images
Conv 32, 3x3, stride=2x2
ReLU
Conv 64, 3x3
ReLU
SeparableConv 128, 3x3
Conv 1x1
stride=2x2
ReLU
SeparableConv 128, 3x3
MaxPooling 3x3, stride=2x2
ReLU
SeparableConv 256, 3x3
Conv 1x1
stride=2x2
ReLU
SeparableConv 256, 3x3
MaxPooling 3x3, stride=2x2
ReLU
SeparableConv 728, 3x3
Conv 1x1
stride=2x2
ReLU
SeparableConv 728, 3x3
MaxPooling 3x3, stride=2x2
19x19x728 feature maps
Middle flow
19x19x728 feature maps
ReLU
SeparableConv 728, 3x3
ReLU
SeparableConv 728, 3x3
ReLU
SeparableConv 728, 3x3
19x19x728 feature maps
Repeated 8 times
Exit flow
19x19x728 feature maps
ReLU
SeparableConv 728, 3x3
Conv 1x1
stride=2x2
ReLU
SeparableConv 1024, 3x3
MaxPooling 3x3, stride=2x2
SeparableConv 1536, 3x3
ReLU
SeparableConv 2048, 3x3
ReLU
GlobalAveragePooling
2048-dimensional vectors
Optional fully-connected
layer(s)
Logistic regression



Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
ResNeXt50	96 MB	0.777	0.938	25,097,128	-
ResNeXt101	170 MB	0.787	0.943	44,315,560	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201
NASNetMobile	23 MB	0.744	0.919	5,326,716	-
NASNetLarge	343 MB	0.825	0.960	88,949,818	-

The top-1 and top-5 accuracy refers to the model's performance on the ImageNet validation dataset.



is it raining

Predicted top-5 answers with confidence:

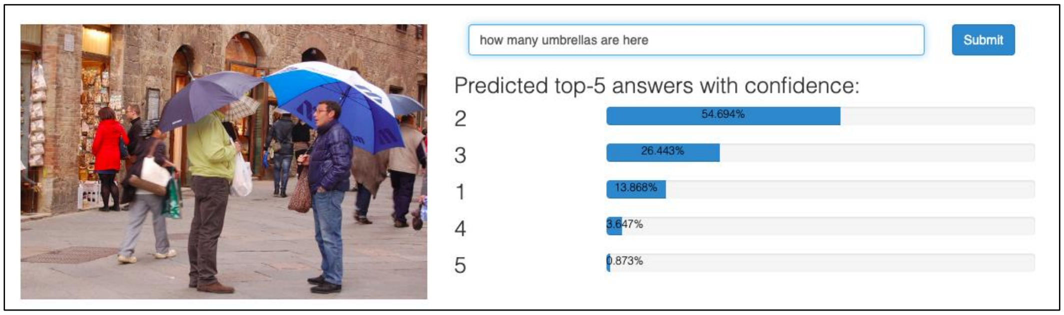
yes 60.228%

no 39.771%

umbrella 0.000%

bag 0.000%

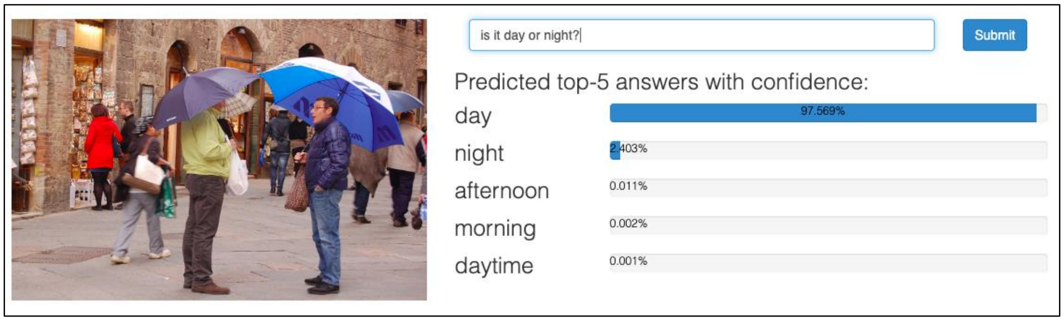
carpet 0.000%



how many umbrellas are here

Predicted top-5 answers with confidence:

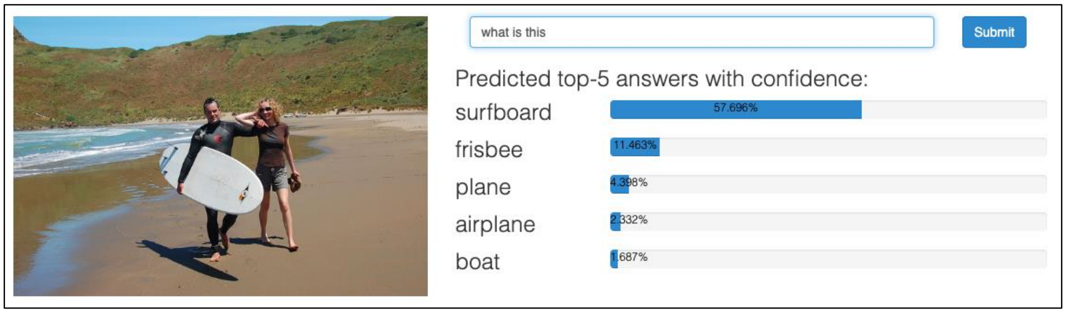
```
2    54.694%
3    26.443%
1    13.868%
4    3.647%
5    0.873%
```



is it day or night?

Predicted top-5 answers with confidence:

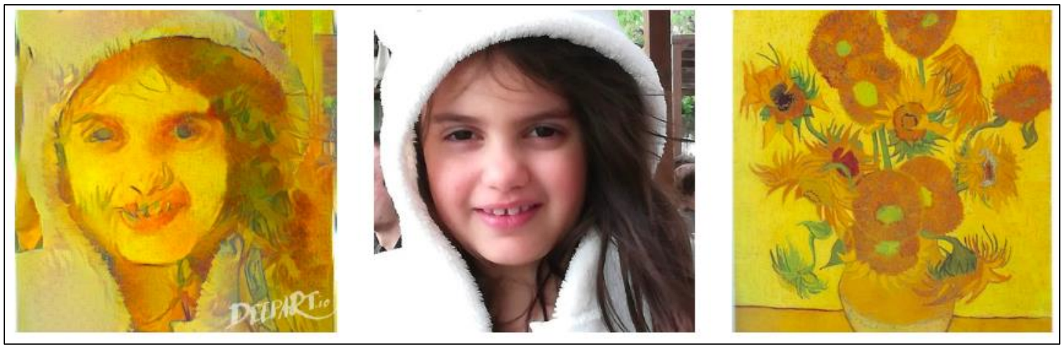
```
day          97.569%
night        2.403%
afternoon    0.011%
morning      0.002%
daytime      0.001%
```

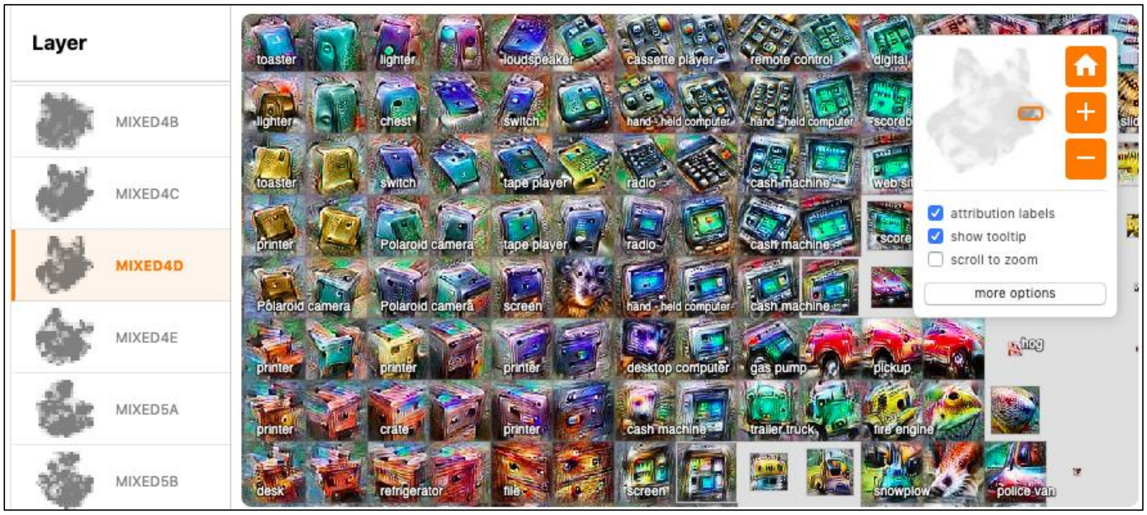


what is this

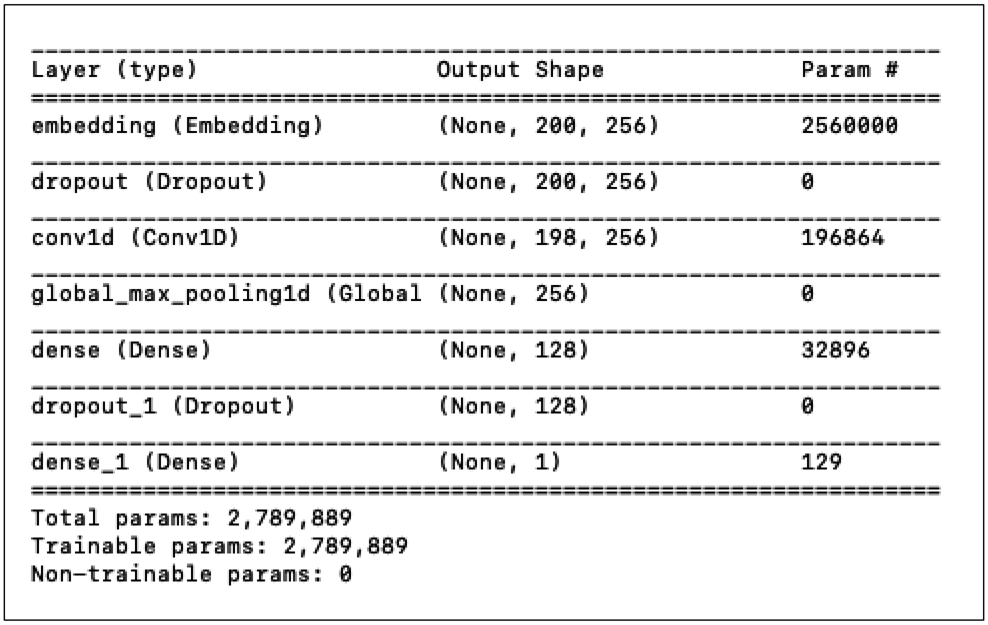
Predicted top-5 answers with confidence:

```
surfboard    57.696%
frisbee      11.463%
plane        4.398%
airplane     2.332%
boat         1.687%
```

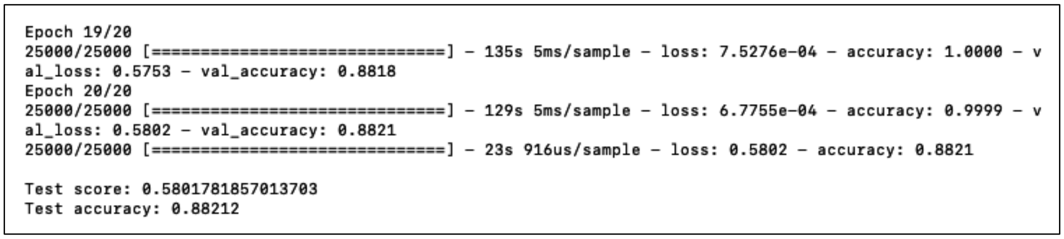




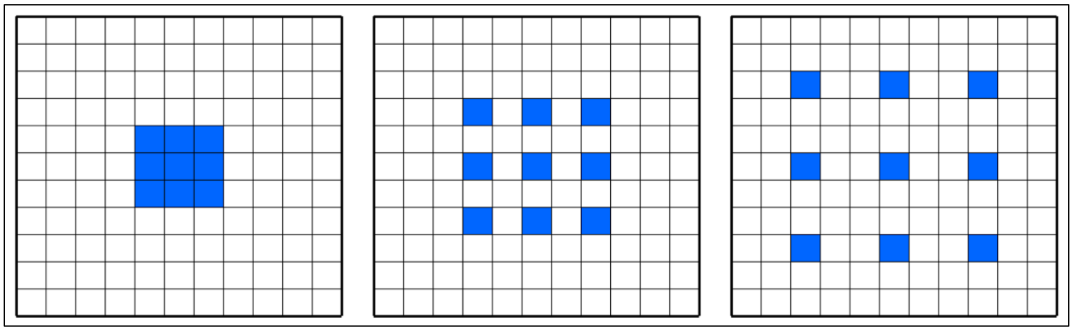
☑ attribution labels
☑ show tooltip
☐ scroll to zoom

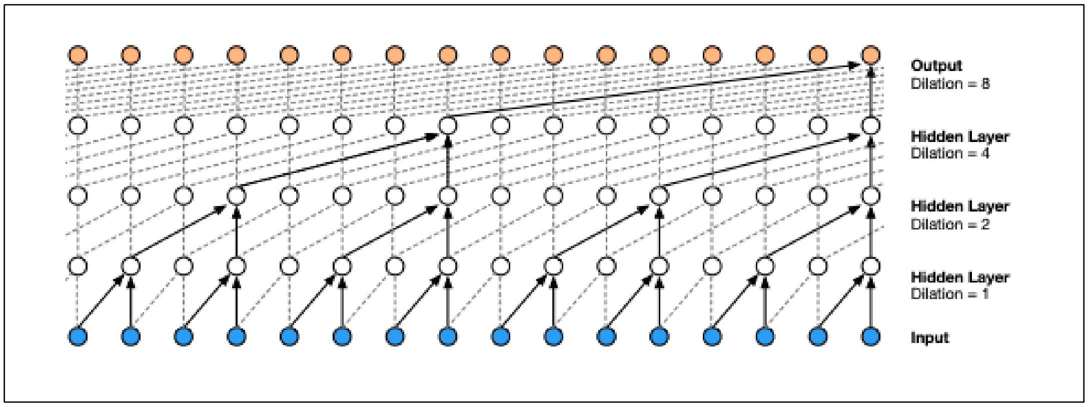


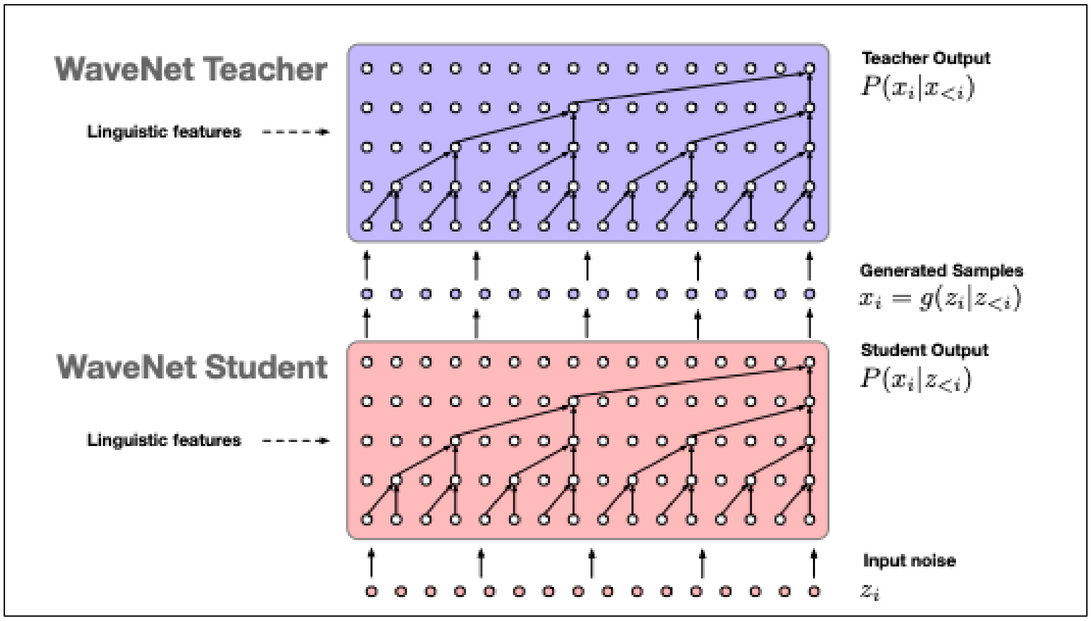
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 200, 256)          2560000
_________________________________________________________________
dropout (Dropout)            (None, 200, 256)          0
_________________________________________________________________
conv1d (Conv1D)              (None, 198, 256)          196864
_________________________________________________________________
global_max_pooling1d (Global (None, 256)               0
_________________________________________________________________
dense (Dense)                (None, 128)               32896
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 129
=================================================================
Total params: 2,789,889
Trainable params: 2,789,889
Non-trainable params: 0
```

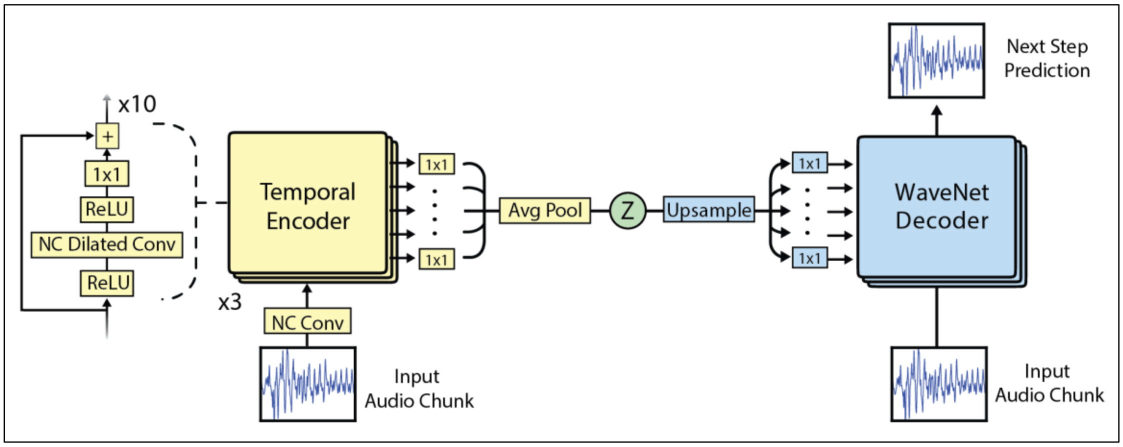


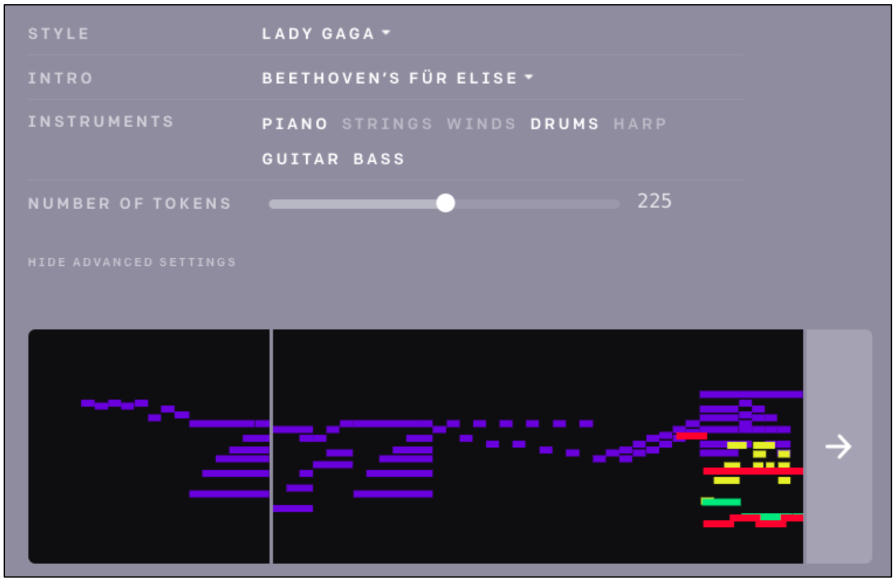
```
Epoch 19/20
25000/25000 [==============================] - 135s 5ms/sample - loss: 7.5276e-04 - accuracy: 1.0000 - v
al_loss: 0.5753 - val_accuracy: 0.8818
Epoch 20/20
25000/25000 [==============================] - 129s 5ms/sample - loss: 6.7755e-04 - accuracy: 0.9999 - v
al_loss: 0.5802 - val_accuracy: 0.8821
25000/25000 [==============================] - 23s 916us/sample - loss: 0.5802 - accuracy: 0.8821

Test score: 0.5801781857013703
Test accuracy: 0.88212
```




Output
Dilation = 8
Hidden Layer
Dilation = 4
Hidden Layer
Dilation = 2
Hidden Layer
Dilation = 1
Input


WaveNet Teacher
Linguistic features
Teacher Output
$P(x_i|x_{<i})$
Generated Samples
$x_i = g(z_i|z_{<i})$
WaveNet Student
Linguistic features
Student Output
$P(x_i|z_{<i})$
Input noise
z_i





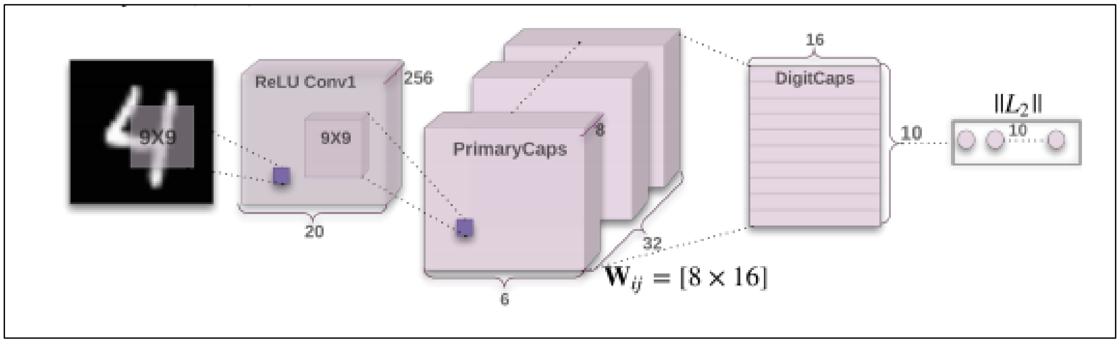
STYLE LADY GAGA

INTRO BEETHOVEN'S FÜR ELISE

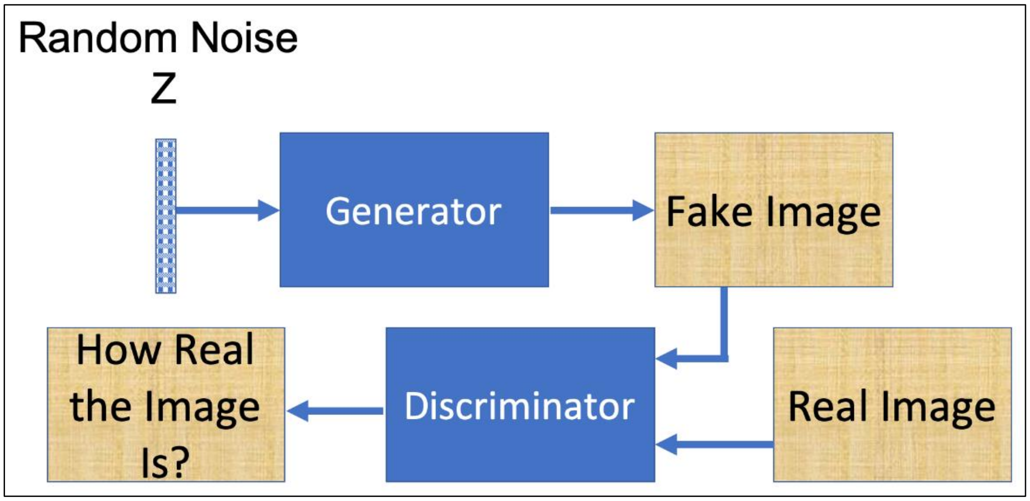
INSTRUMENTS PIANO STRINGS WINDS DRUMS HARP GUITAR BASS

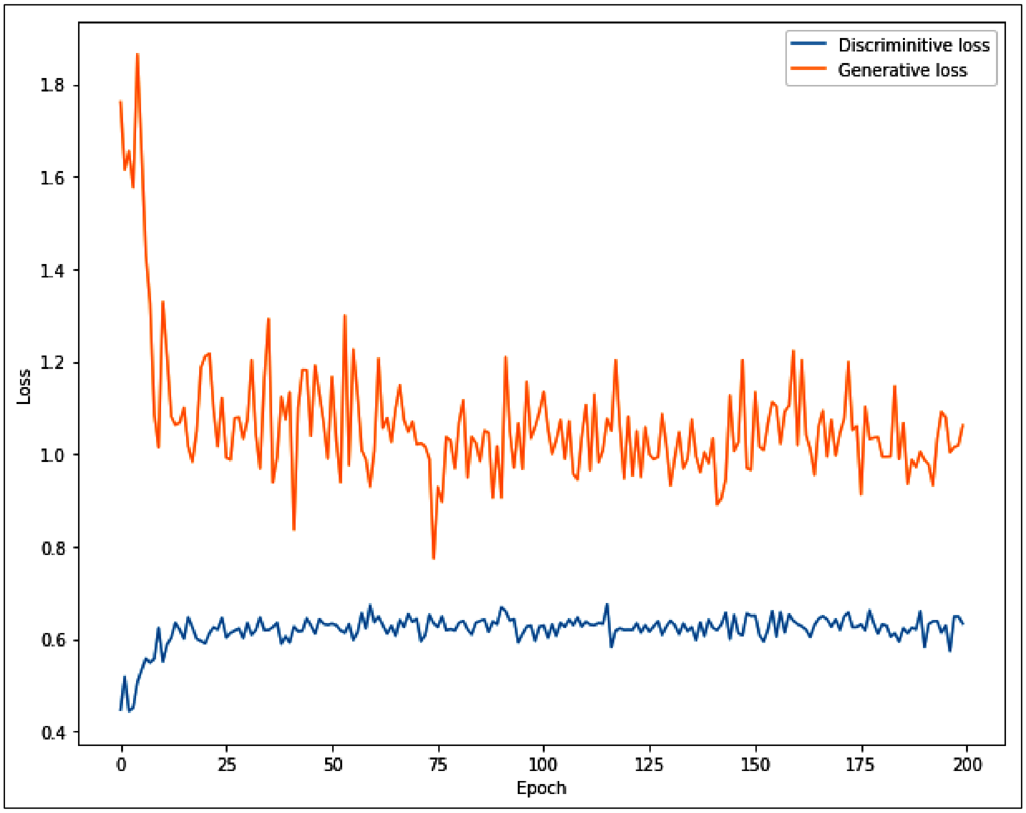
NUMBER OF TOKENS 225

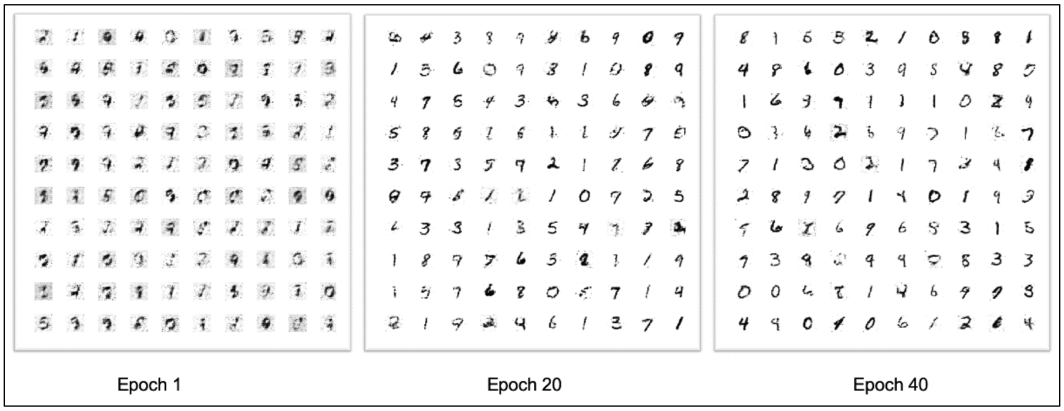
HIDE ADVANCED SETTINGS



Chapter 6: Generative Adversarial Networks



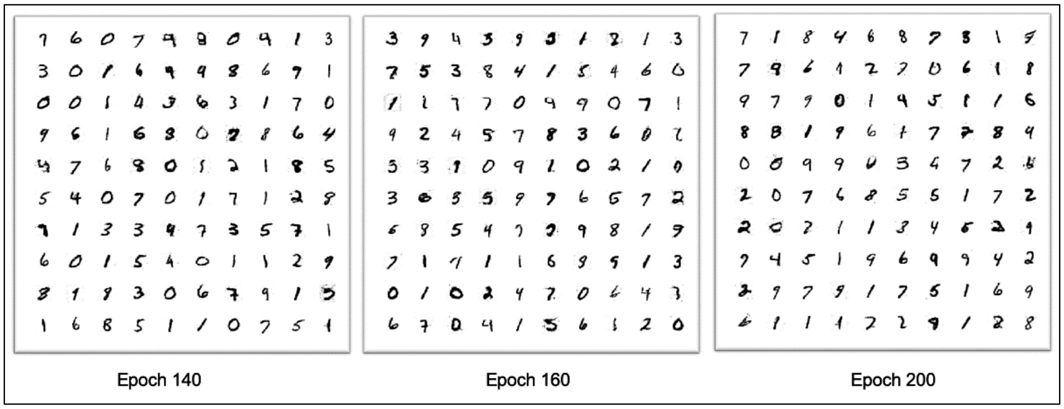




Epoch 1

Epoch 20

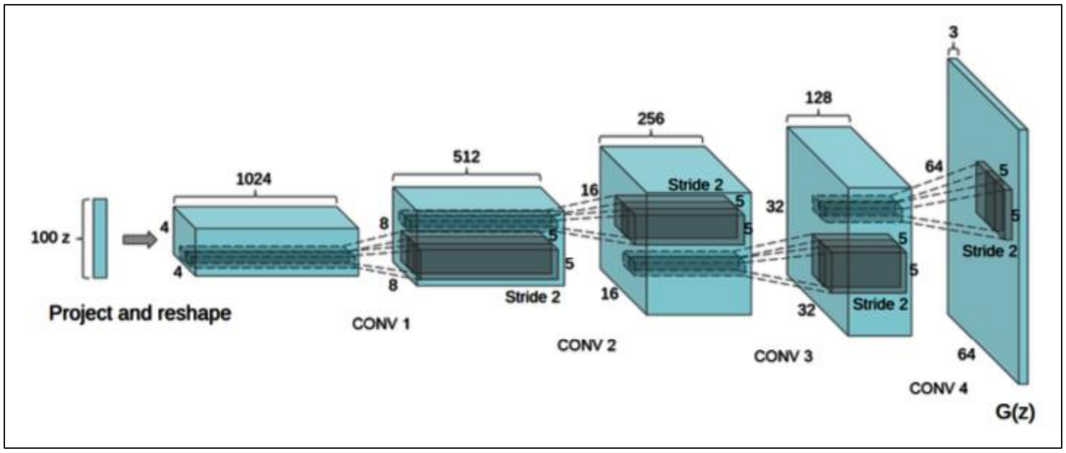
Epoch 40

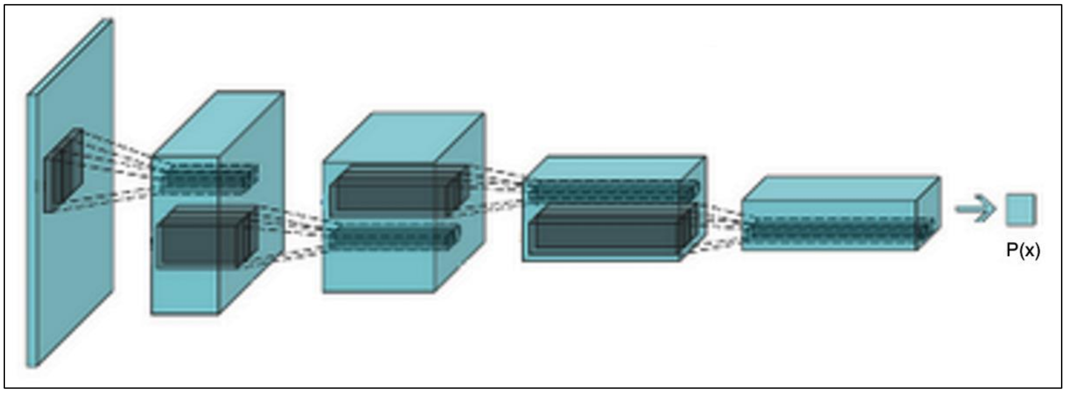


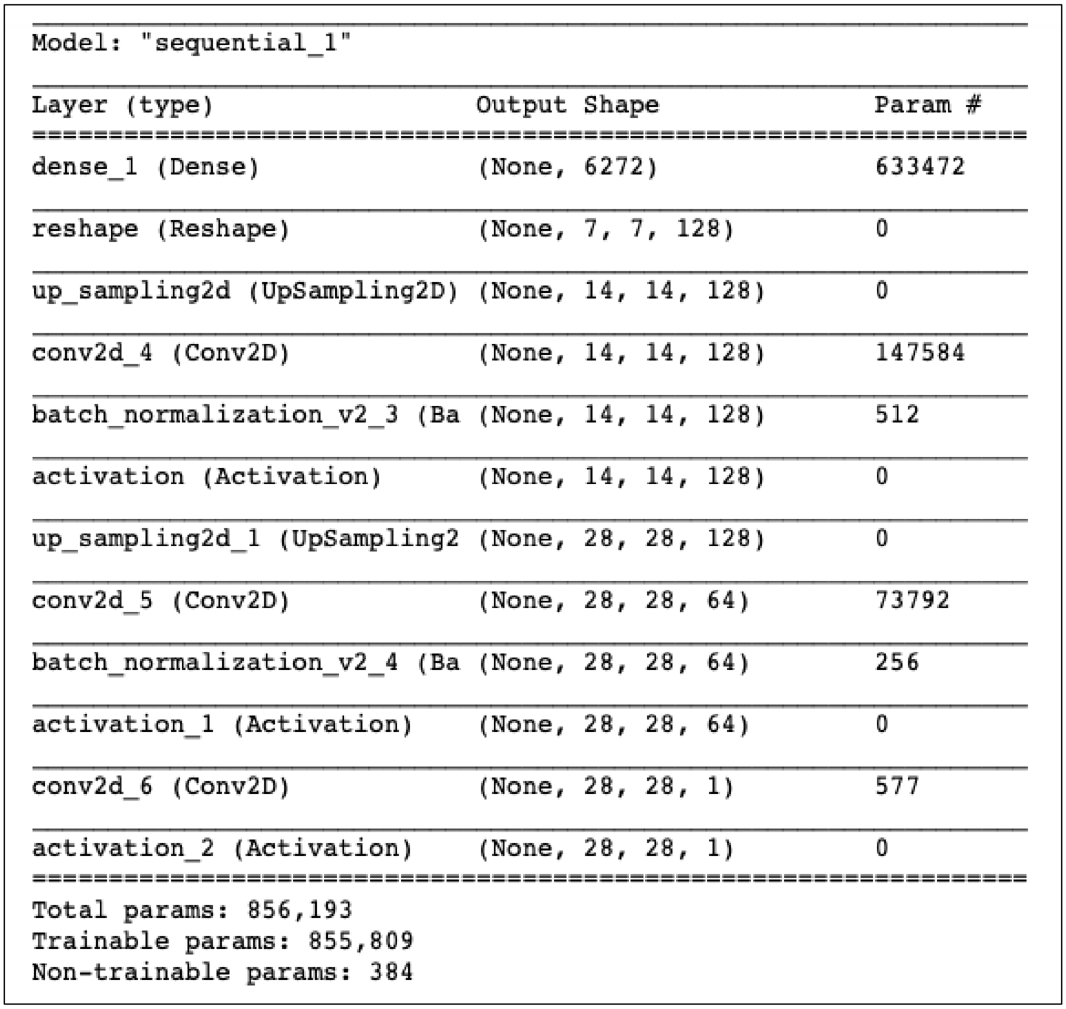
Epoch 140

Epoch 160

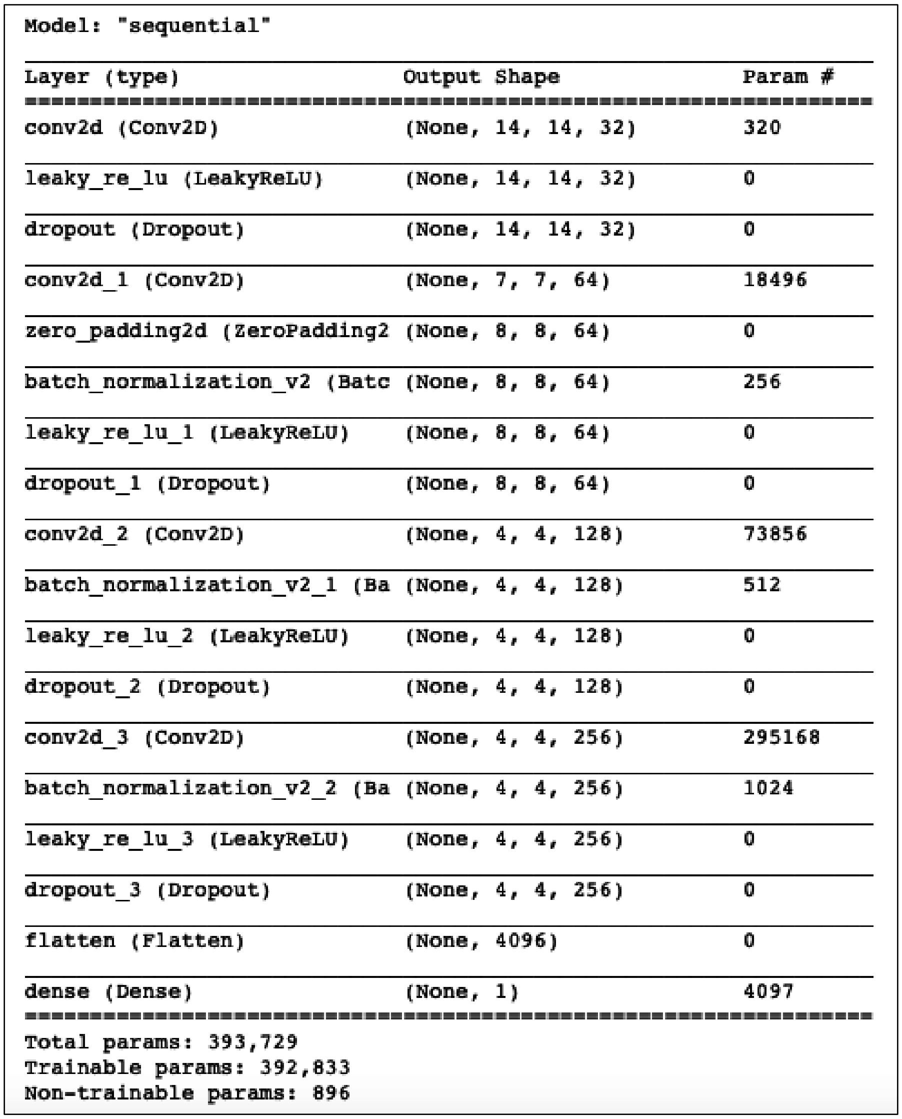
Epoch 200



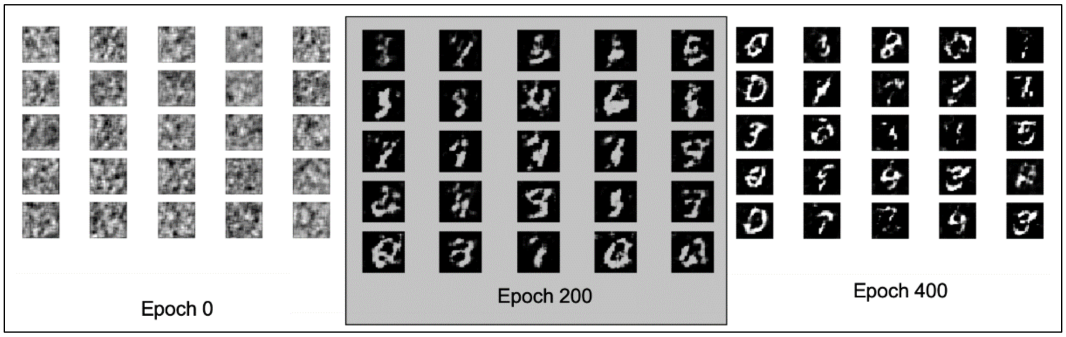


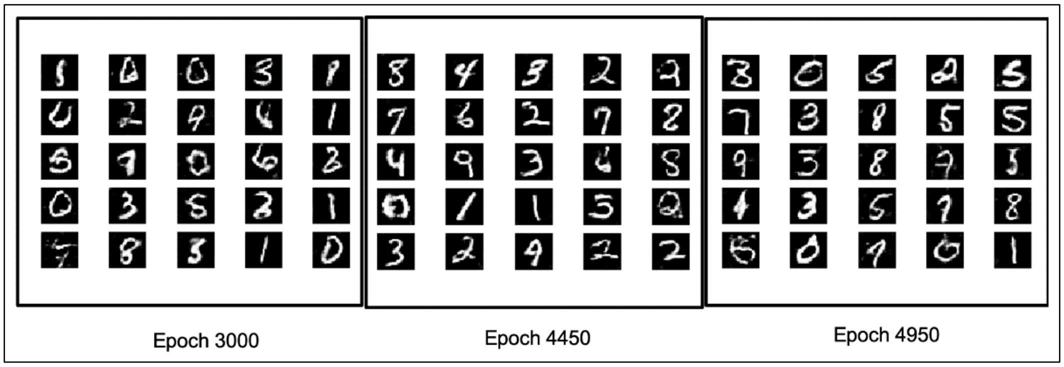


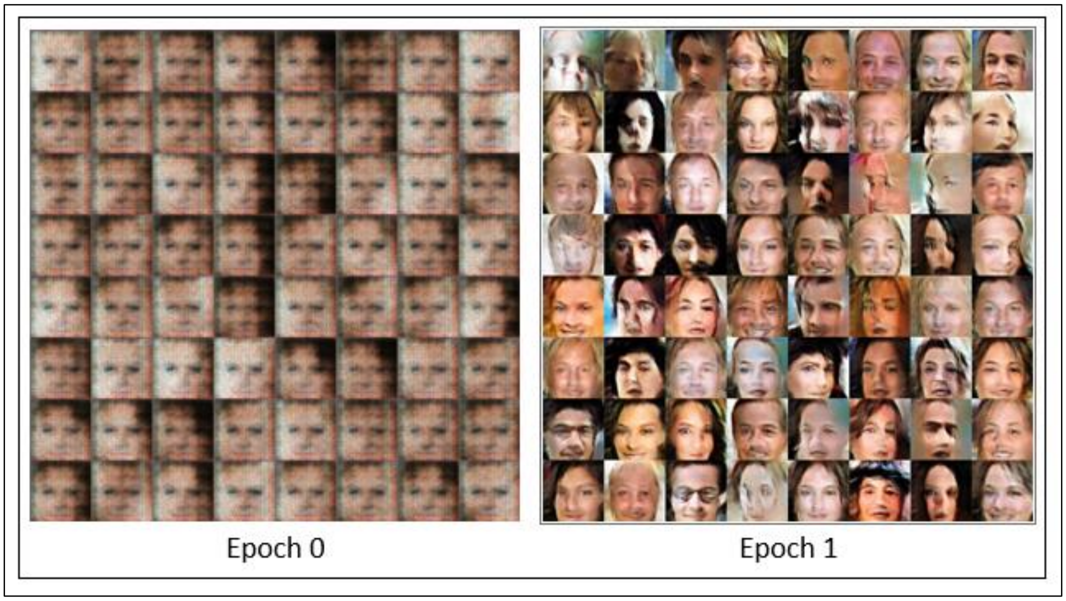
```
_________________________________________________________________
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 6272)              633472
_________________________________________________________________
reshape (Reshape)            (None, 7, 7, 128)         0
_________________________________________________________________
up_sampling2d (UpSampling2D) (None, 14, 14, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 14, 14, 128)       147584
_________________________________________________________________
batch_normalization_v2_3 (Ba (None, 14, 14, 128)       512
_________________________________________________________________
activation (Activation)      (None, 14, 14, 128)       0
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 28, 28, 128)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 28, 28, 64)        73792
_________________________________________________________________
batch_normalization_v2_4 (Ba (None, 28, 28, 64)        256
_________________________________________________________________
activation_1 (Activation)    (None, 28, 28, 64)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 28, 28, 1)         577
_________________________________________________________________
activation_2 (Activation)    (None, 28, 28, 1)         0
=================================================================
Total params: 856,193
Trainable params: 855,809
Non-trainable params: 384
```

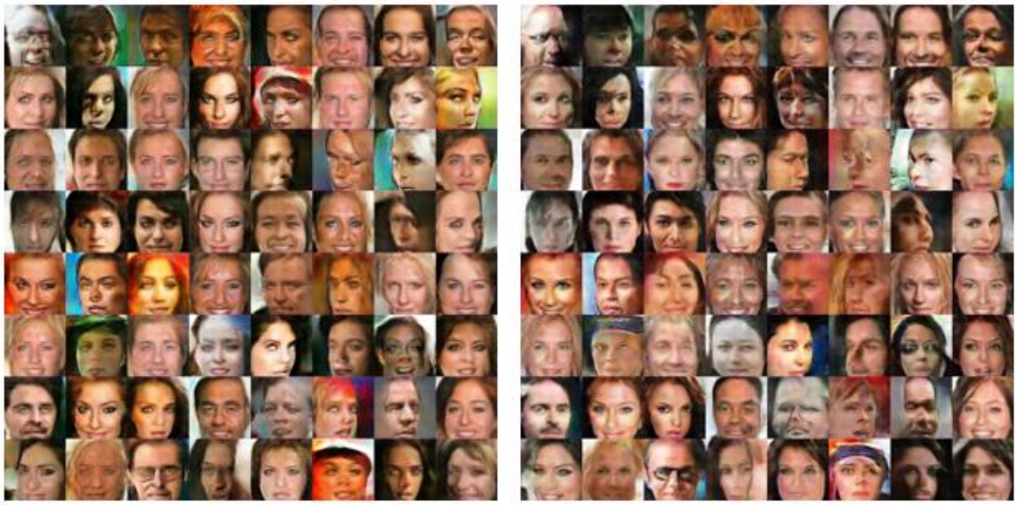


```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 14, 14, 32)        320
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 14, 14, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 7, 7, 64)          18496
_________________________________________________________________
zero_padding2d (ZeroPadding2 (None, 8, 8, 64)          0
_________________________________________________________________
batch_normalization_v2 (Batc (None, 8, 8, 64)          256
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 8, 8, 64)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 4, 128)         73856
_________________________________________________________________
batch_normalization_v2_1 (Ba (None, 4, 4, 128)         512
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 4, 4, 128)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 4, 4, 128)         0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 4, 4, 256)         295168
_________________________________________________________________
batch_normalization_v2_2 (Ba (None, 4, 4, 256)         1024
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 4, 4, 256)         0
_________________________________________________________________
dropout_3 (Dropout)          (None, 4, 4, 256)         0
_________________________________________________________________
flatten (Flatten)            (None, 4096)              0
_________________________________________________________________
dense (Dense)                (None, 1)                 4097
=================================================================
Total params: 393,729
Trainable params: 392,833
Non-trainable params: 896
```


Epoch 0
Epoch 200
Epoch 400

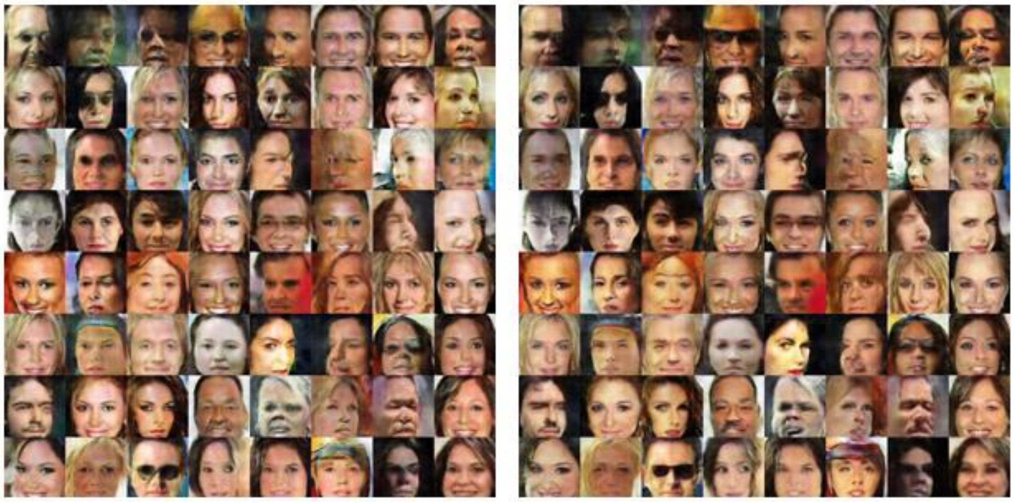

Epoch 3000
Epoch 4450
Epoch 4950


Epoch 0
Epoch 1



Epoch 2

Epoch 3



Epoch 4

Epoch 5

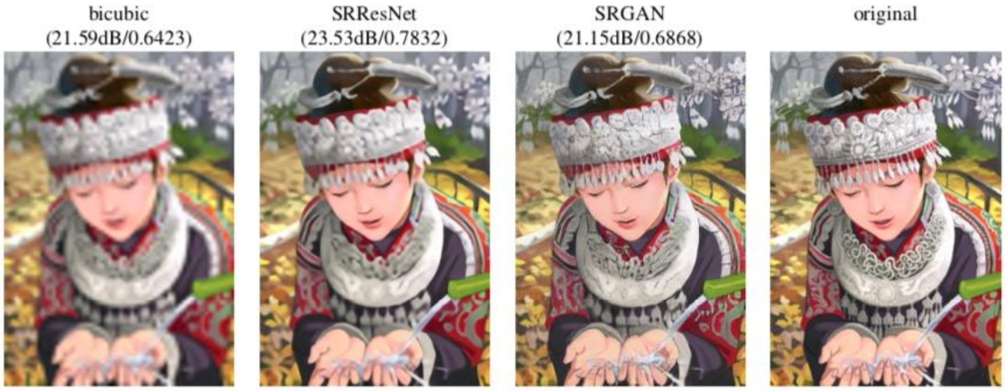
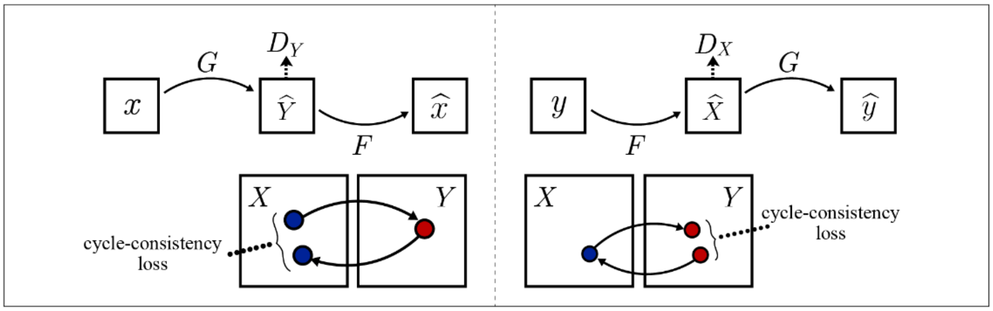
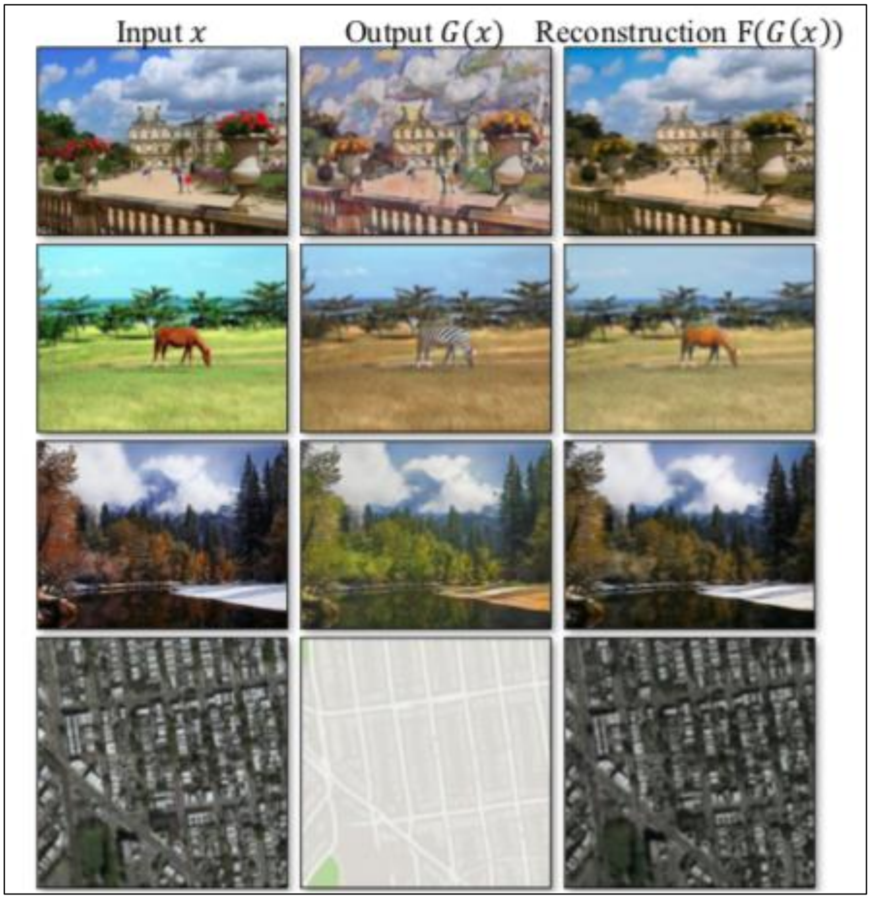
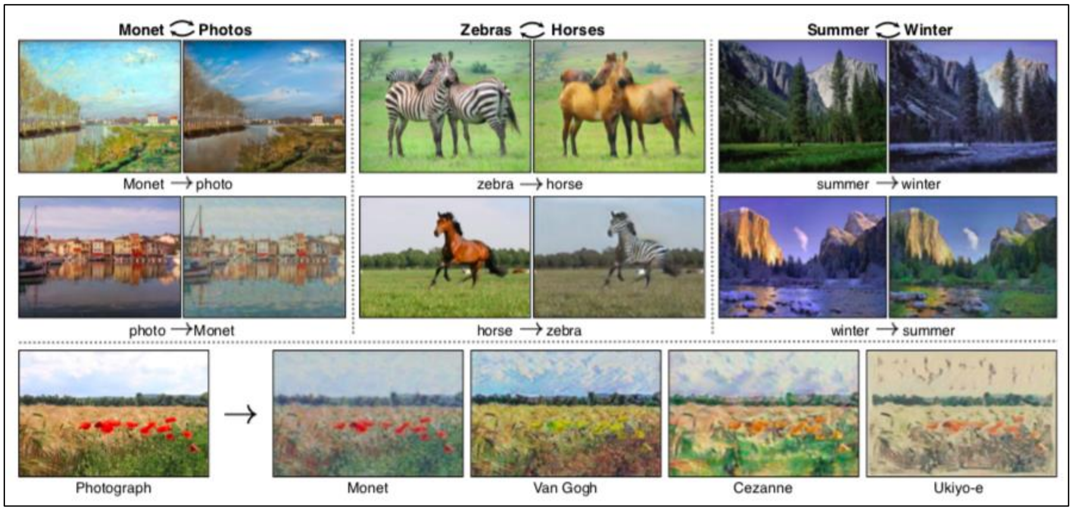
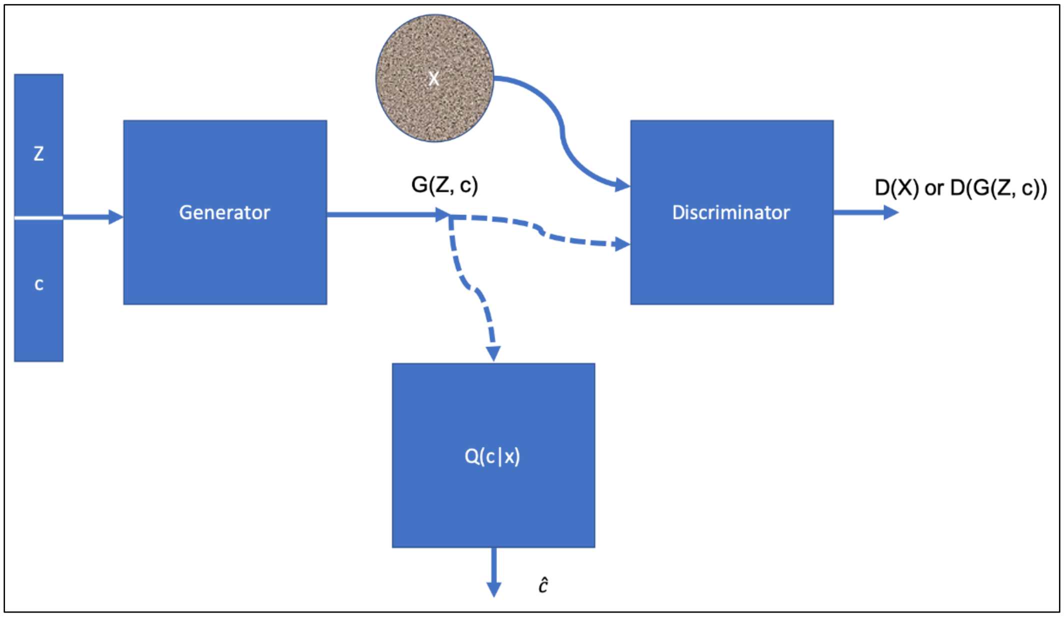


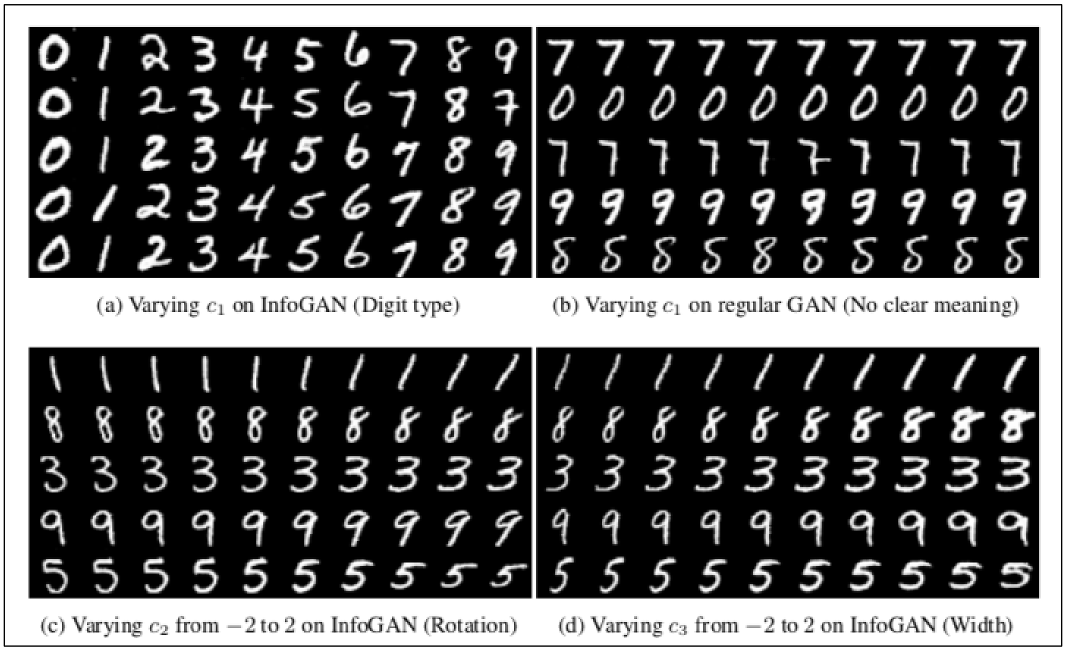
Figure 2: From left to right: bicubic interpolation, deep residual network optimized for MSE, deep residual generative adversarial network optimized for a loss more sensitive to human perception, original HR image. Corresponding PSNR and SSIM are shown in brackets. [4× upscaling]




Input x
Output $G(x)$
Reconstruction $F(G(x))$


Monet ⇄ Photos
Zebras ⇄ Horses
Summer ⇄ Winter
Monet → photo
zebra → horse
summer → winter
photo → Monet
horse → zebra
winter → summer
Photograph
→
Monet
Van Gogh
Cezanne
Ukiyo-e



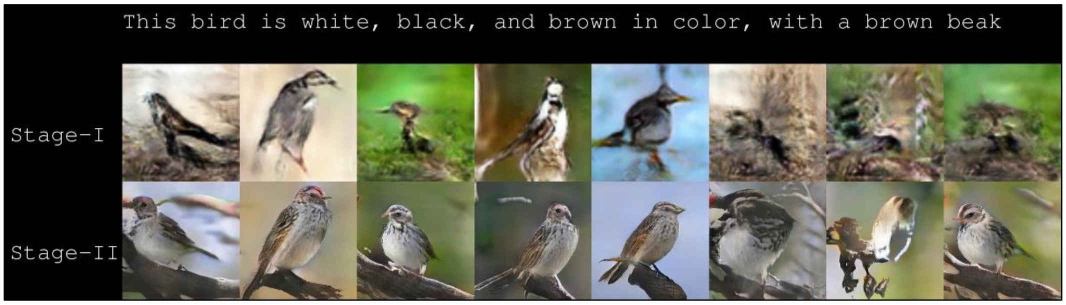


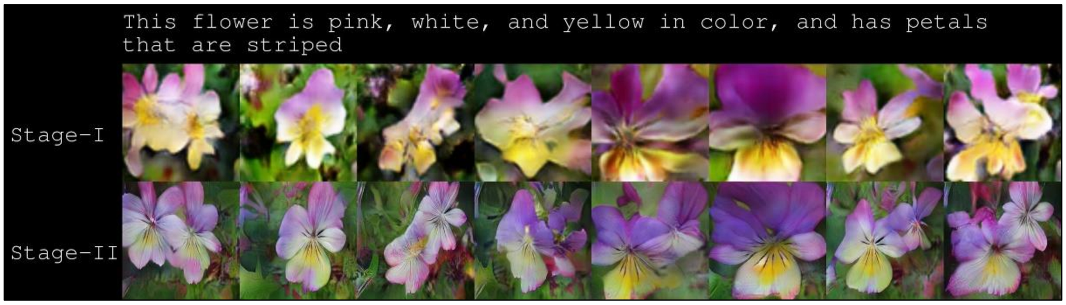
(a) Varying c_1 on InfoGAN (Digit type)

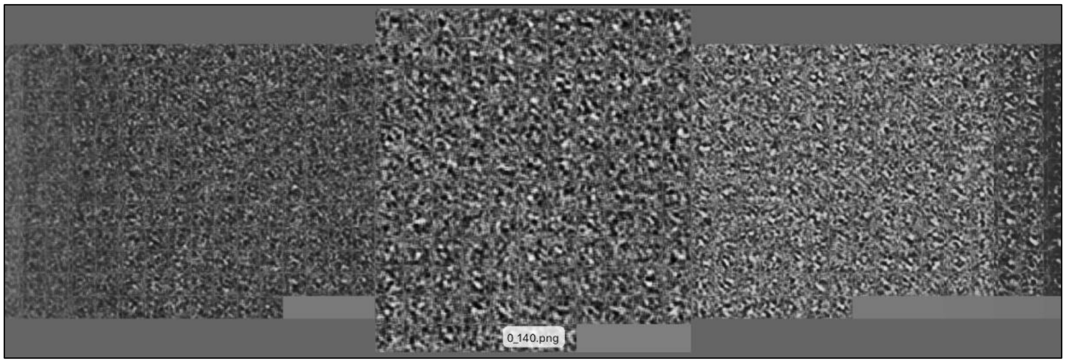
(b) Varying c_1 on regular GAN (No clear meaning)

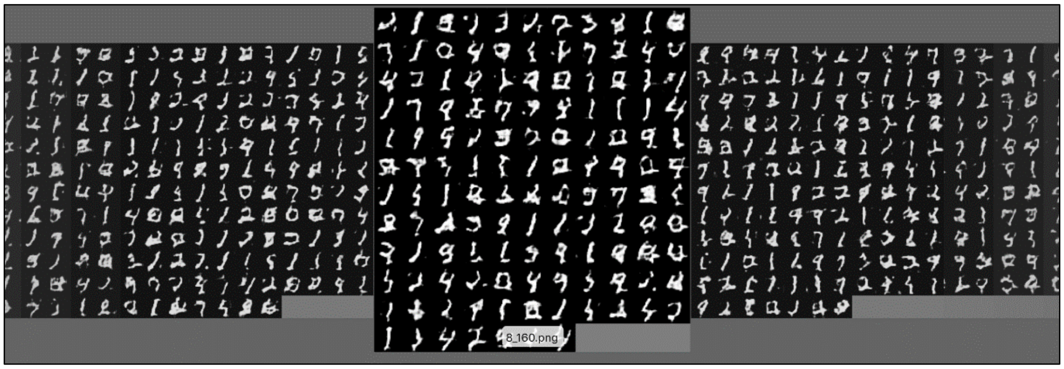
(c) Varying c_2 from -2 to 2 on InfoGAN (Rotation)

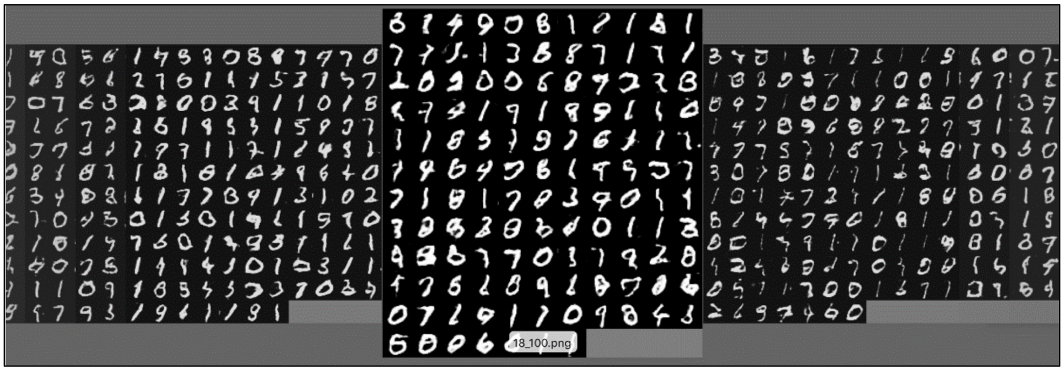
(d) Varying c_3 from -2 to 2 on InfoGAN (Width)

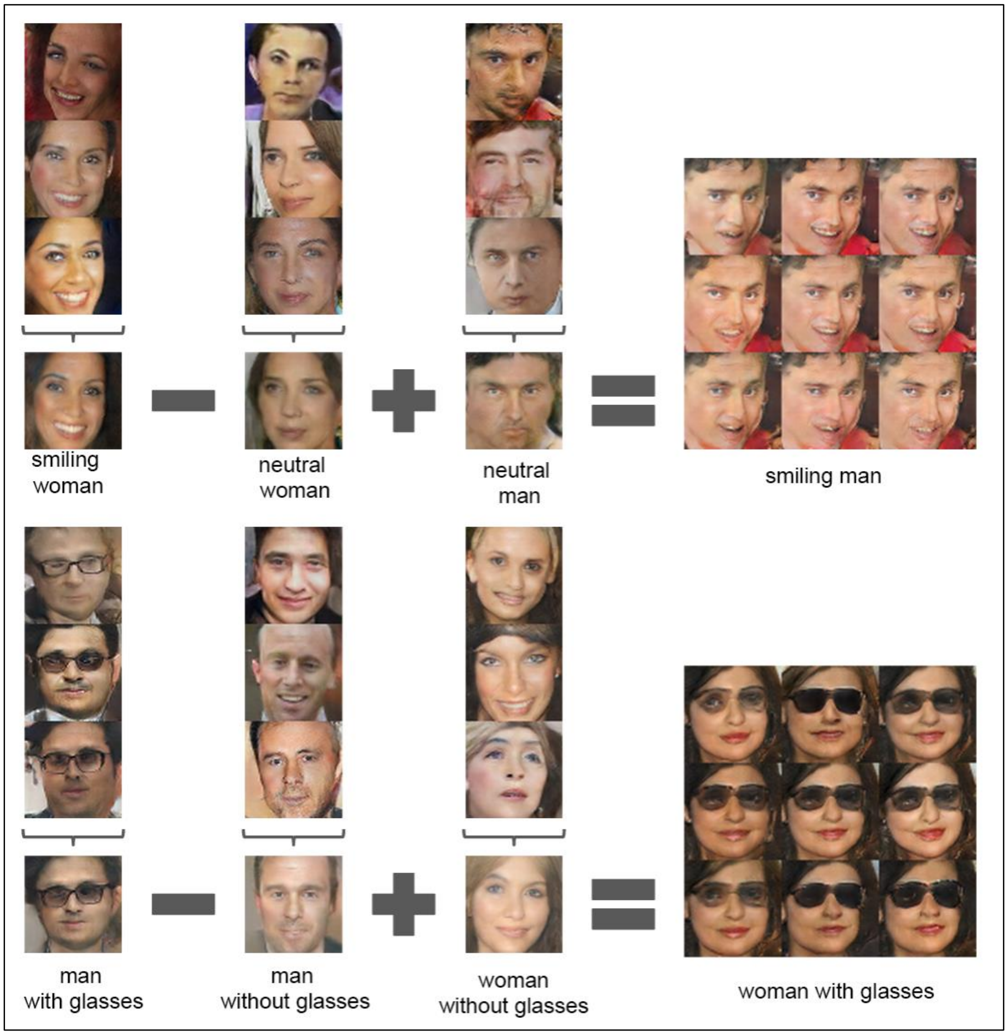

This bird is white, black, and brown in color, with a brown beak
Stage-I
Stage-II

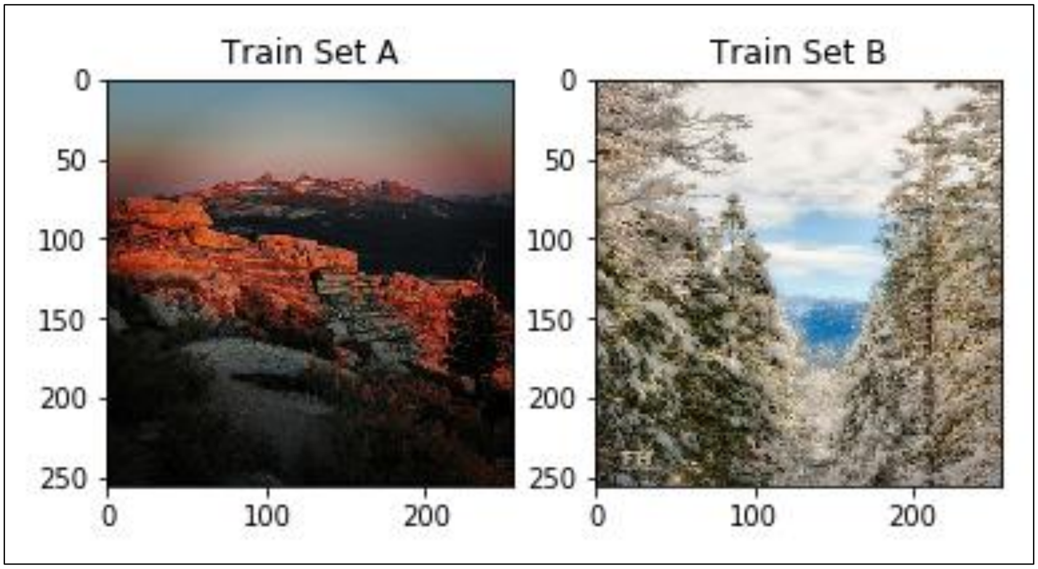

This flower is pink, white, and yellow in color, and has petals that are striped
Stage-I
Stage-II


0_140.png


8_160.png

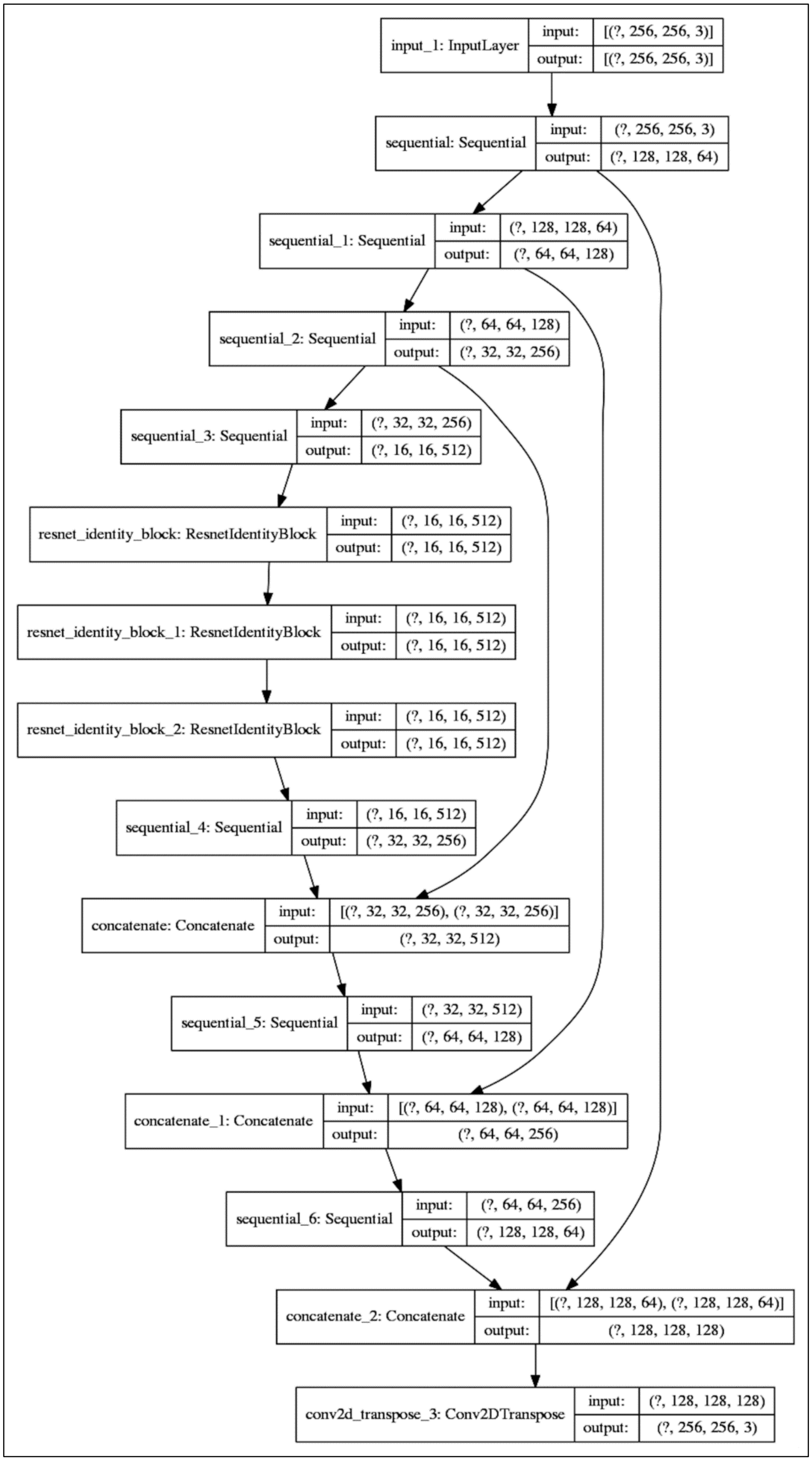

18_100.png

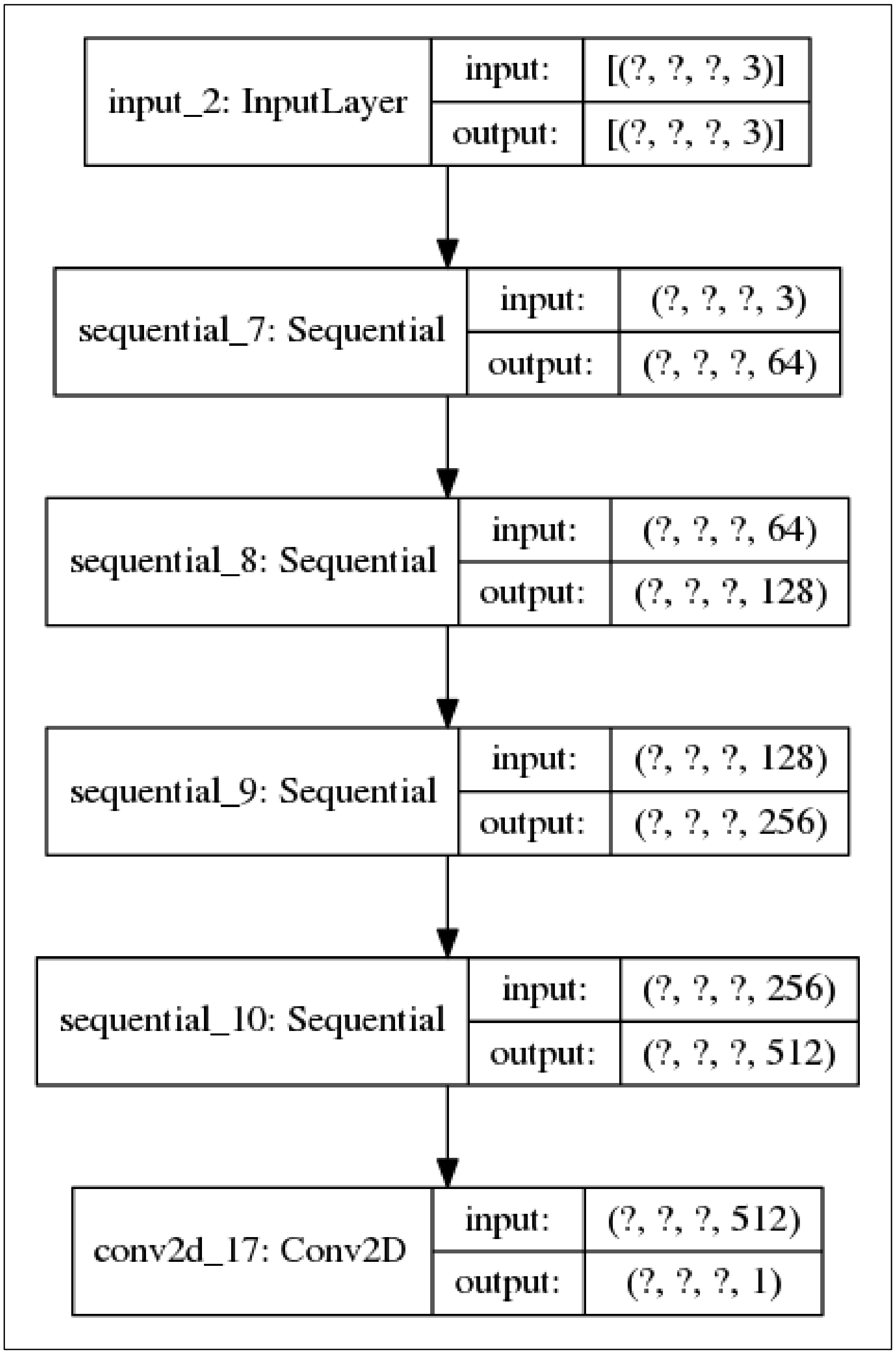

smiling woman
neutral woman
neutral man
smiling man
man with glasses
man without glasses
woman without glasses
woman with glasses

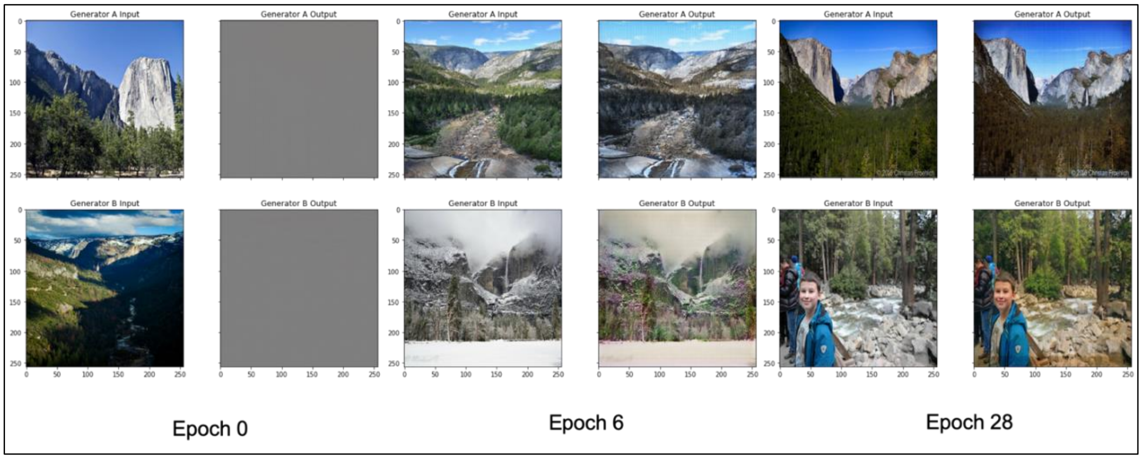


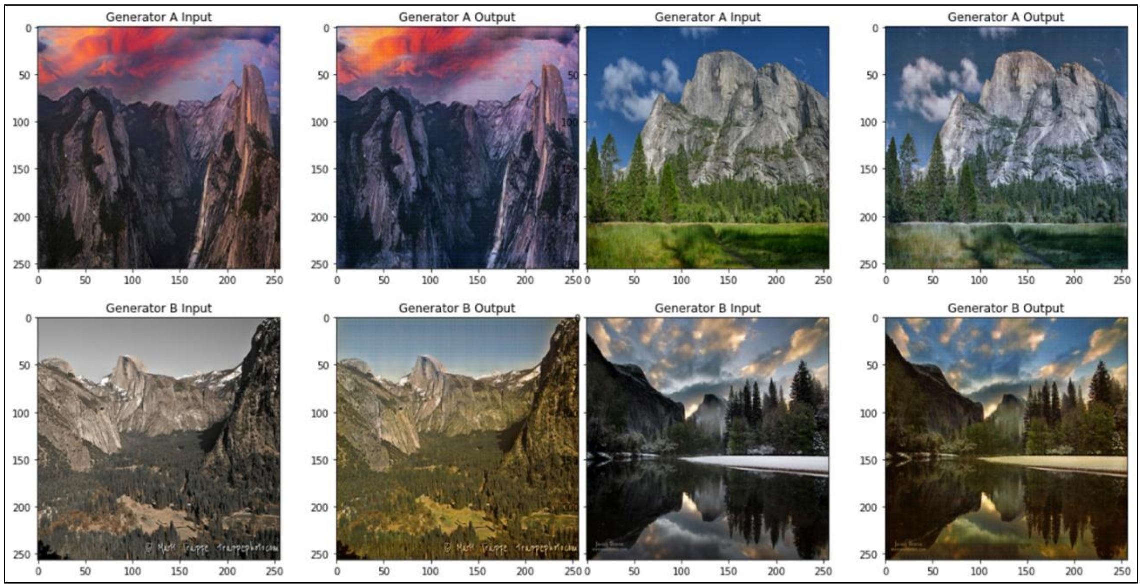
Train Set A

Train Set B

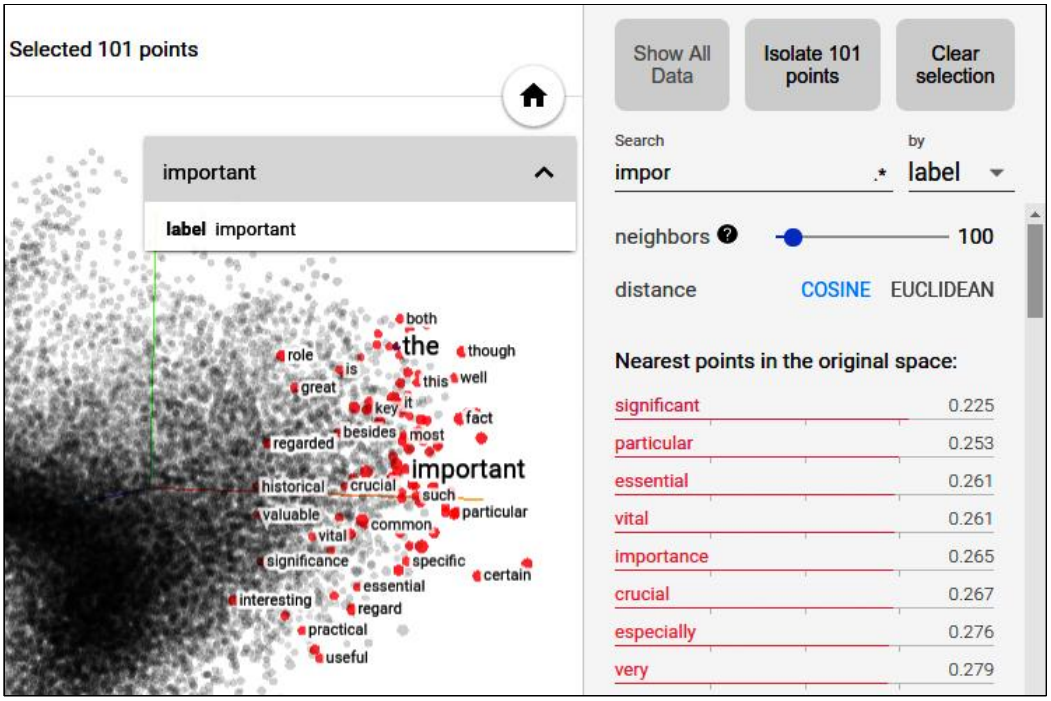

input_1: InputLayer
input: [(?, 256, 256, 3)]
output: [(?, 256, 256, 3)]
sequential: Sequential
input: (?, 256, 256, 3)
output: (?, 128, 128, 64)
sequential_1: Sequential
input: (?, 128, 128, 64)
output: (?, 64, 64, 128)
sequential_2: Sequential
input: (?, 64, 64, 128)
output: (?, 32, 32, 256)
sequential_3: Sequential
input: (?, 32, 32, 256)
output: (?, 16, 16, 512)
resnet_identity_block: ResnetIdentityBlock
input: (?, 16, 16, 512)
output: (?, 16, 16, 512)
resnet_identity_block_1: ResnetIdentityBlock
input: (?, 16, 16, 512)
output: (?, 16, 16, 512)
resnet_identity_block_2: ResnetIdentityBlock
input: (?, 16, 16, 512)
output: (?, 16, 16, 512)
sequential_4: Sequential
input: (?, 16, 16, 512)
output: (?, 32, 32, 256)
concatenate: Concatenate
input: [(?, 32, 32, 256), (?, 32, 32, 256)]
output: (?, 32, 32, 512)
sequential_5: Sequential
input: (?, 32, 32, 512)
output: (?, 64, 64, 128)
concatenate_1: Concatenate
input: [(?, 64, 64, 128), (?, 64, 64, 128)]
output: (?, 64, 64, 256)
sequential_6: Sequential
input: (?, 64, 64, 256)
output: (?, 128, 128, 64)
concatenate_2: Concatenate
input: [(?, 128, 128, 64), (?, 128, 128, 64)]
output: (?, 128, 128, 128)
conv2d_transpose_3: Conv2DTranspose
input: (?, 128, 128, 128)
output: (?, 256, 256, 3)


input_2: InputLayer
input:
[(?, ?, ?, 3)]
output:
[(?, ?, ?, 3)]
sequential_7: Sequential
input:
(?, ?, ?, 3)
output:
(?, ?, ?, 64)
sequential_8: Sequential
input:
(?, ?, ?, 64)
output:
(?, ?, ?, 128)
sequential_9: Sequential
input:
(?, ?, ?, 128)
output:
(?, ?, ?, 256)
sequential_10: Sequential
input:
(?, ?, ?, 256)
output:
(?, ?, ?, 512)
conv2d_17: Conv2D
input:
(?, ?, ?, 512)
output:
(?, ?, ?, 1)


Generator A Input
Generator A Output
Generator A Input
Generator A Output
Generator A Input
Generator A Output
Generator B Input
Generator B Output
Generator B Input
Generator B Output
Generator B Input
Generator B Output
0
50
100
150
200
250
Epoch 0
Epoch 6
Epoch 28


Generator A Input
Generator A Output
Generator A Input
Generator A Output
Generator B Input
Generator B Output
Generator B Input
Generator B Output
0
50
100
150
200
250

Chapter 7: Word Embeddings



Selected 101 points

Show All Data | Isolate 101 points | Clear selection

important

label important

Search: impor

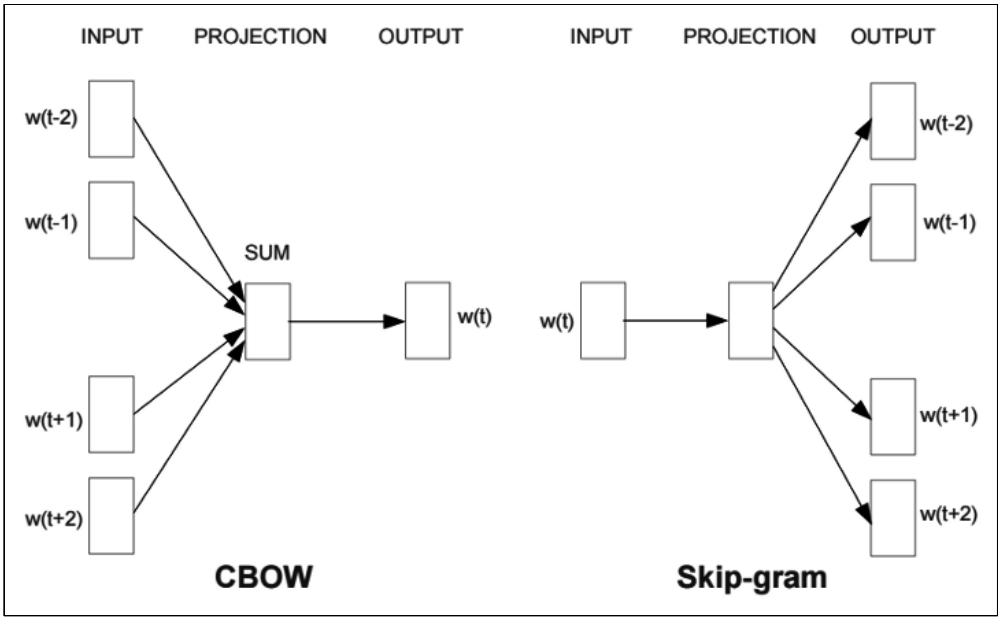
by: label

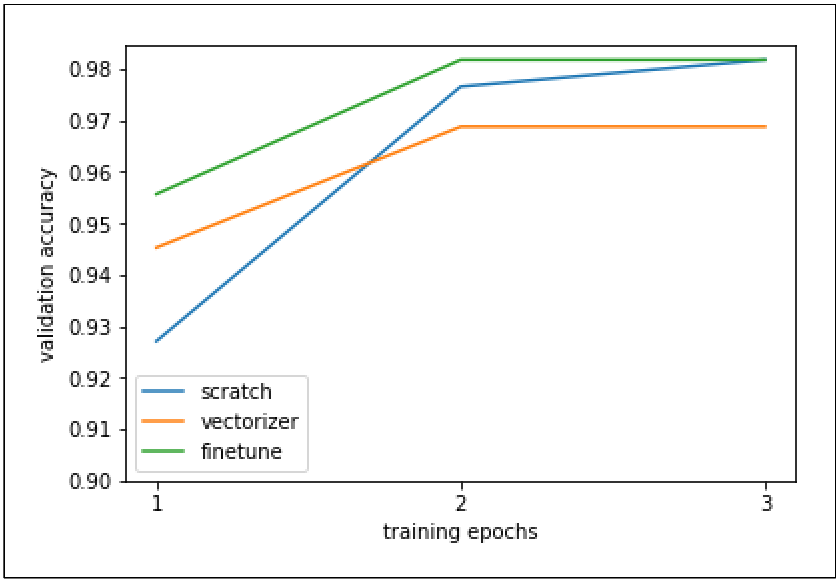
neighbors: 100

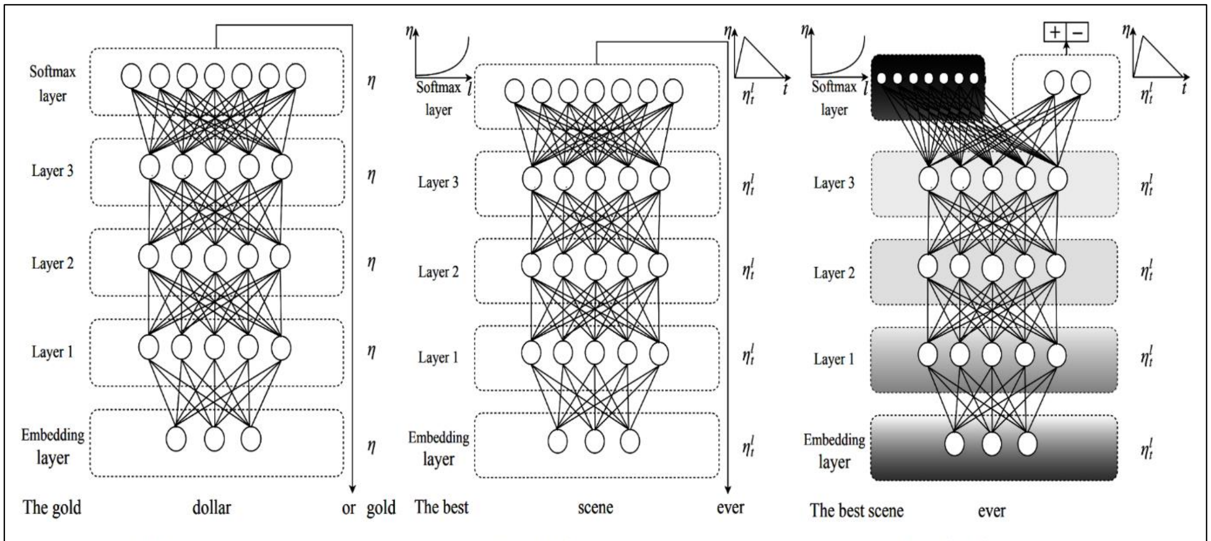
distance: COSINE EUCLIDEAN

Nearest points in the original space:

significant	0.225
particular	0.253
essential	0.261
vital	0.261
importance	0.265
crucial	0.267
especially	0.276
very	0.279





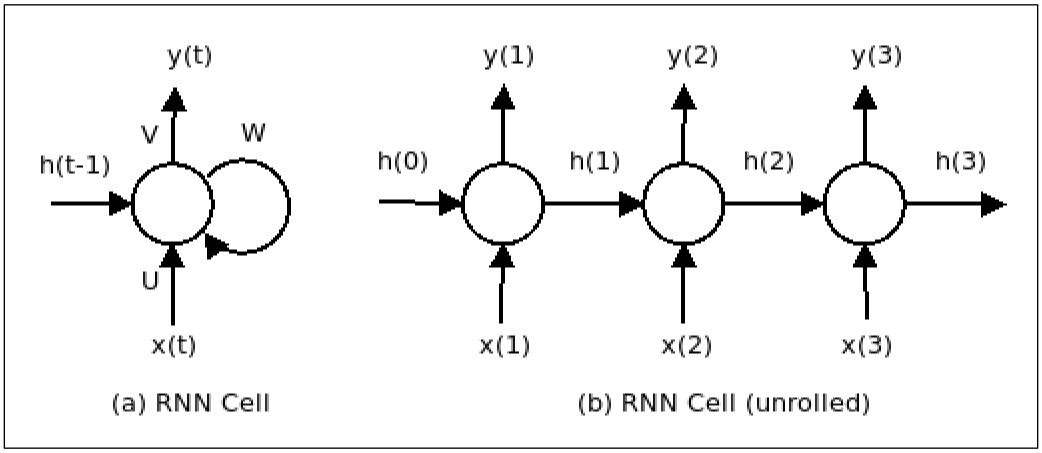


(a) LM pre-training

(b) LM fine-tuning

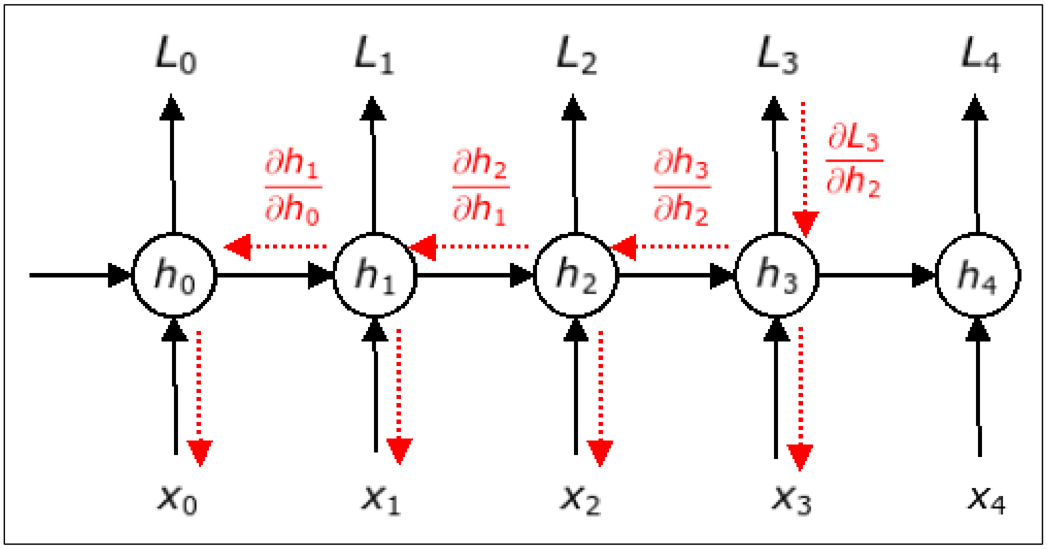
(c) Classifier fine-tuning

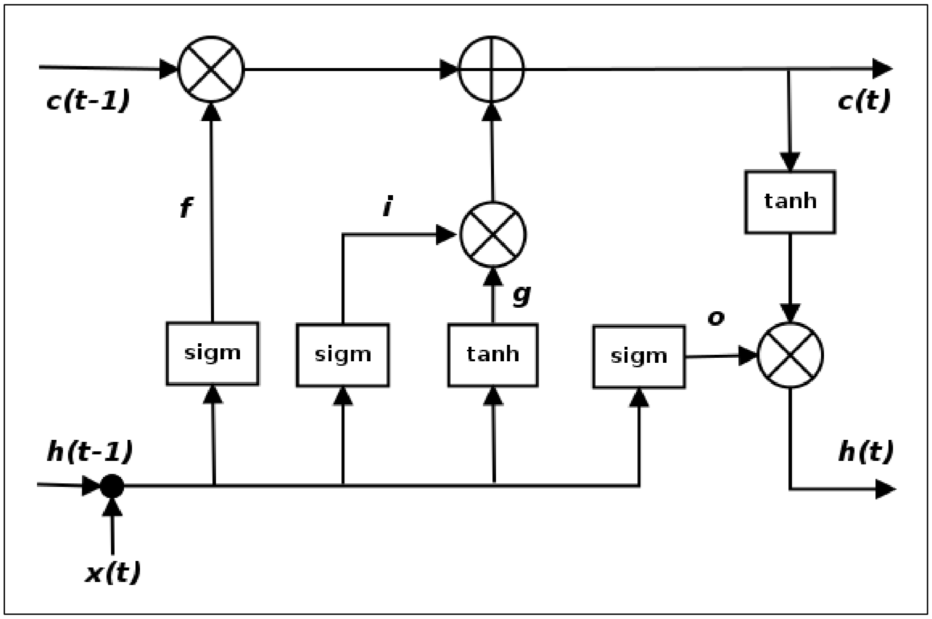
Chapter 8: Recurrent Neural Networks

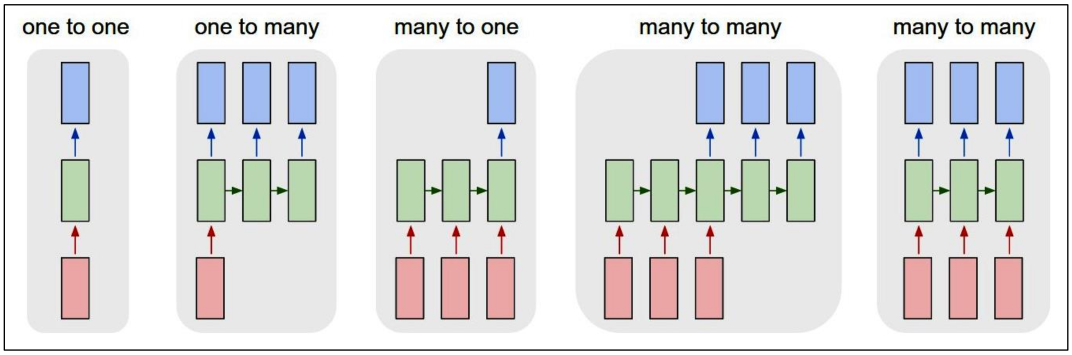


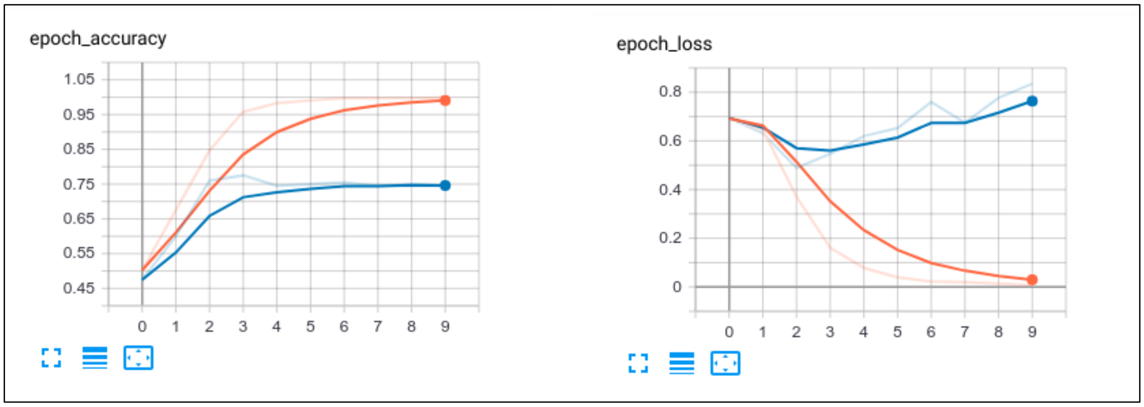
(a) RNN Cell

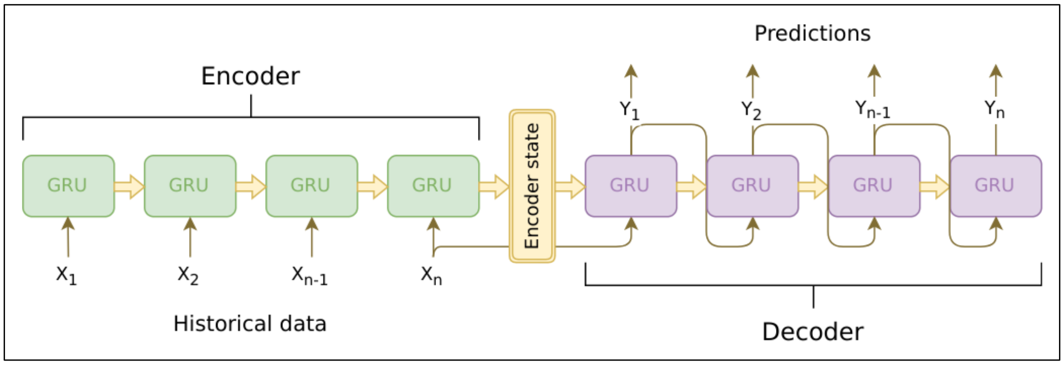
(b) RNN Cell (unrolled)

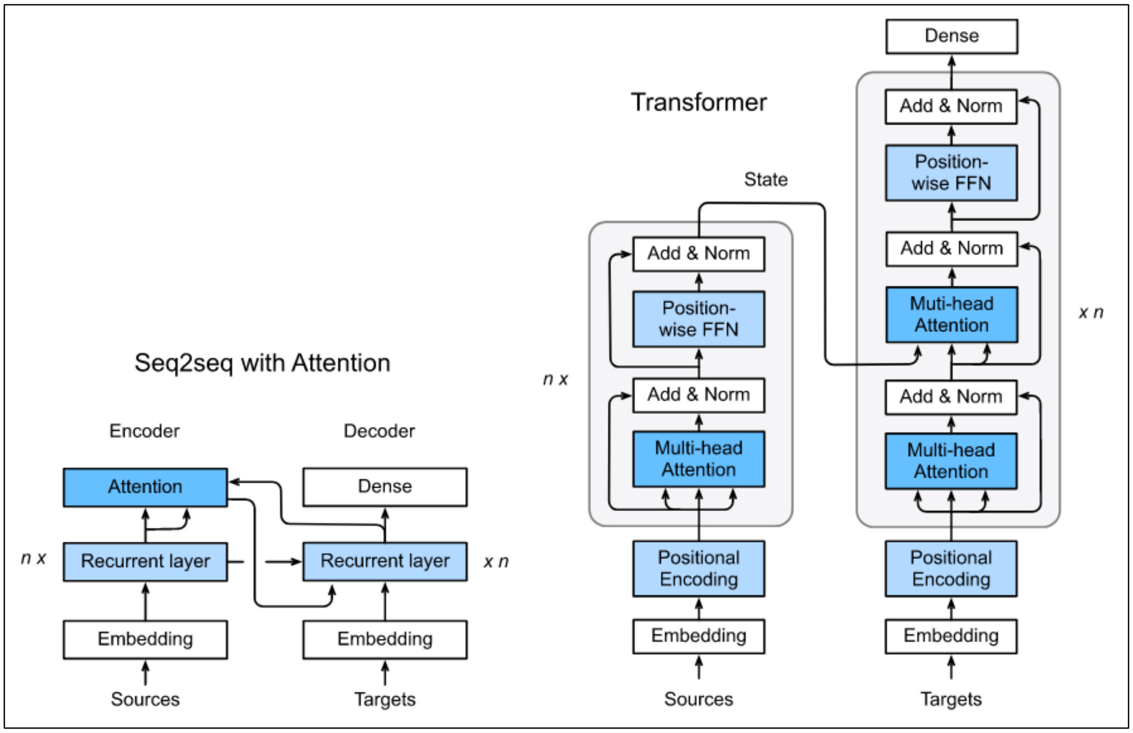

L_0
L_1
L_2
L_3
L_4
$\frac{\partial h_1}{\partial h_0}$
$\frac{\partial h_2}{\partial h_1}$
$\frac{\partial h_3}{\partial h_2}$
$\frac{\partial L_3}{\partial h_2}$
h_0
h_1
h_2
h_3
h_4
x_0
x_1
x_2
x_3
x_4


c(t-1)
c(t)
f
i
g
o
sigm
sigm
tanh
sigm
tanh
h(t-1)
h(t)
x(t)

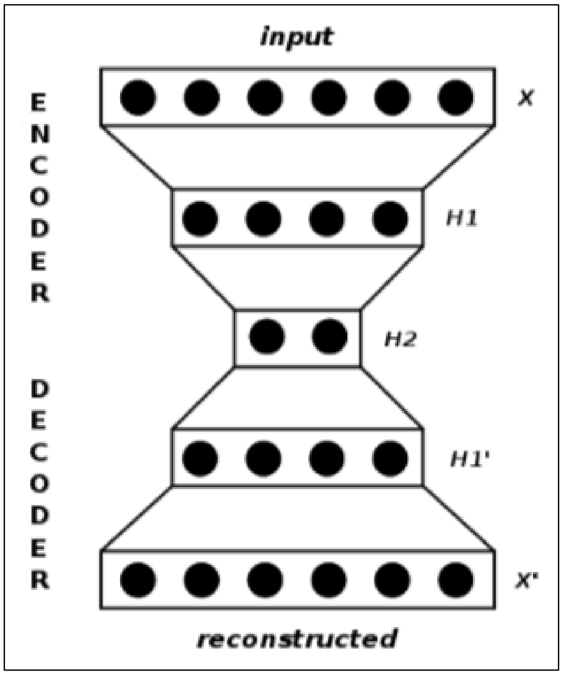

one to one
one to many
many to one
many to many
many to many

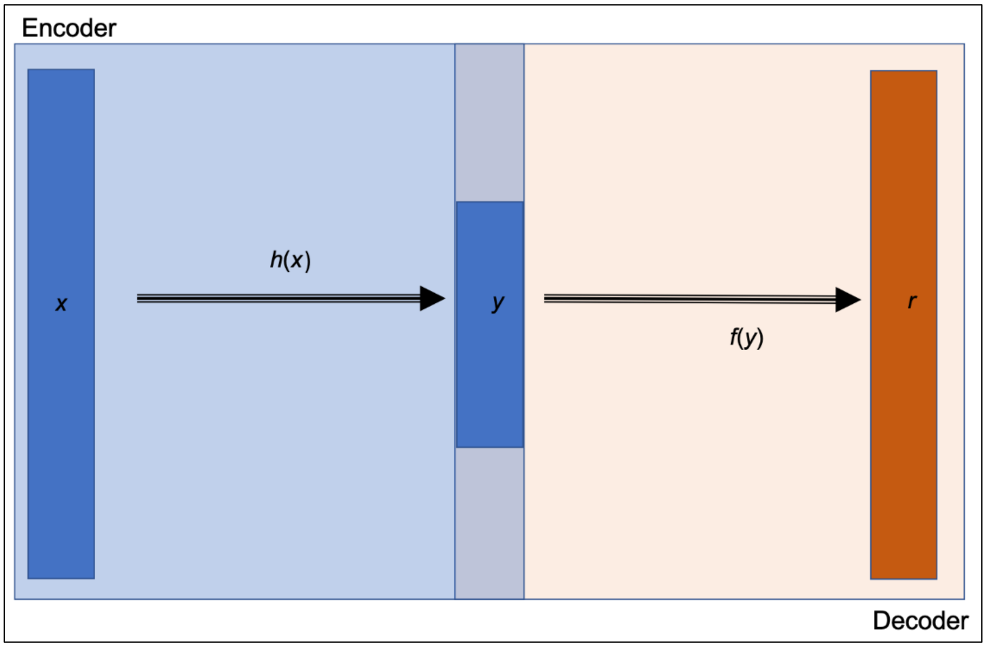

epoch_accuracy
1.05
0.95
0.85
0.75
0.65
0.55
0.45
0
1
2
3
4
5
6
7
8
9
epoch_loss
0.8
0.6
0.4
0.2
0
0
1
2
3
4
5
6
7
8
9

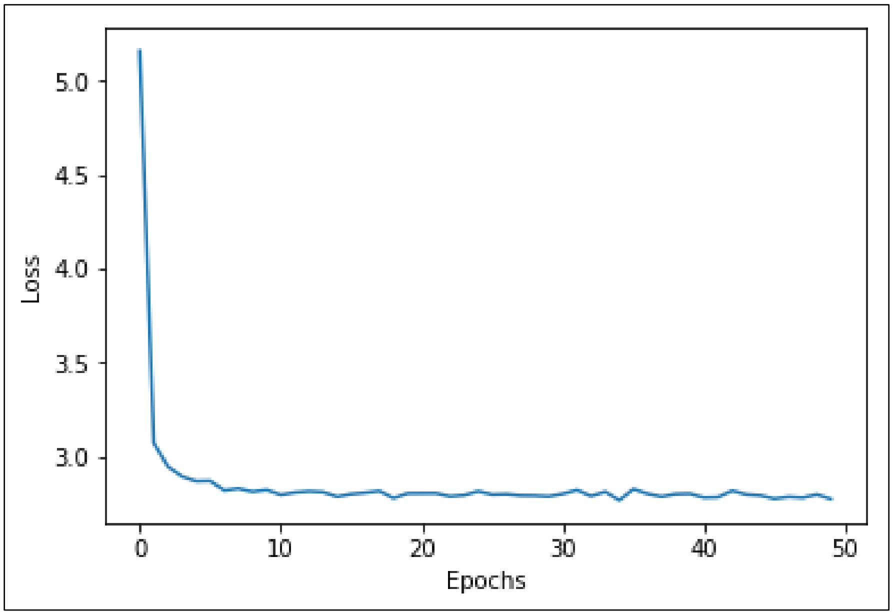

Predictions
Encoder
Y_1
Y_2
Y_{n-1}
Y_n
GRU
GRU
GRU
GRU
Encoder state
GRU
GRU
GRU
GRU
X_1
X_2
X_{n-1}
X_n
Historical data
Decoder

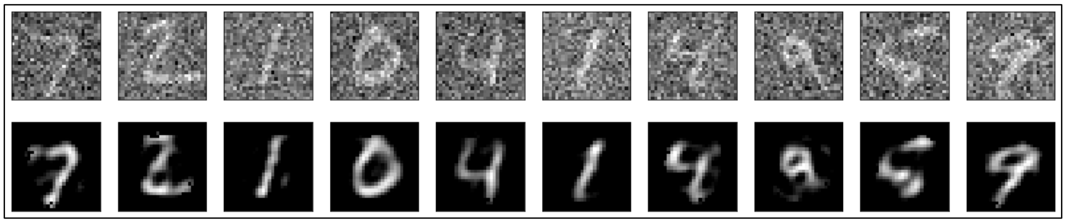

Dense
Transformer
Add & Norm
Position-wise FFN
State
Add & Norm
Add & Norm
Position-wise FFN
Muti-head Attention
x n
Seq2seq with Attention
n x
Add & Norm
Add & Norm
Multi-head Attention
Multi-head Attention
Encoder
Decoder
Attention
Dense
n x
Recurrent layer
Recurrent layer
x n
Positional Encoding
Positional Encoding
Embedding
Embedding
Embedding
Embedding
Sources
Targets
Sources
Targets

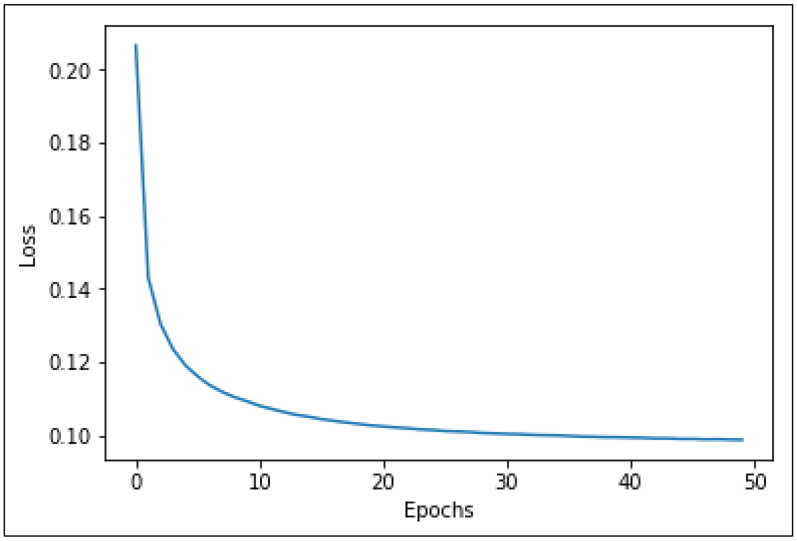
Chapter 9: Autoencoders

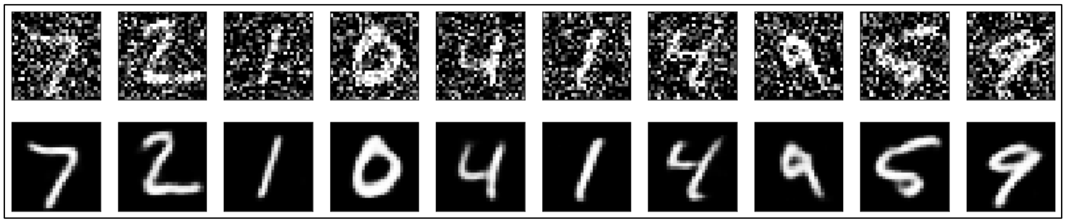


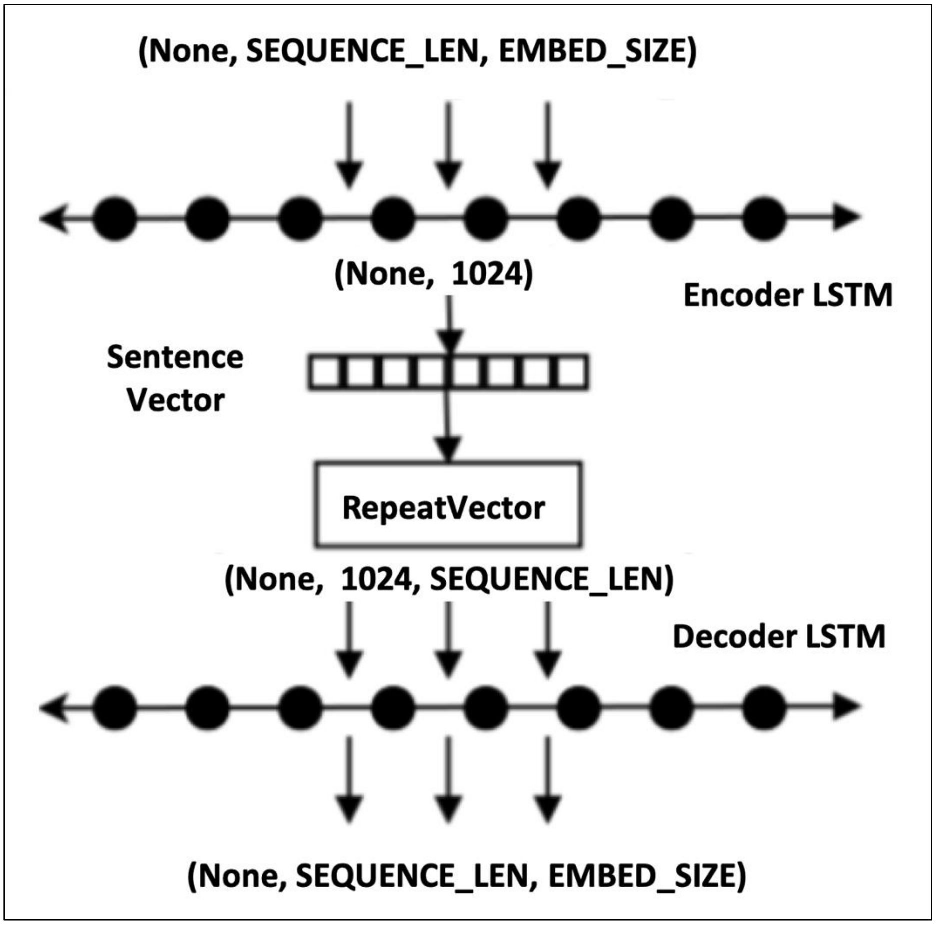


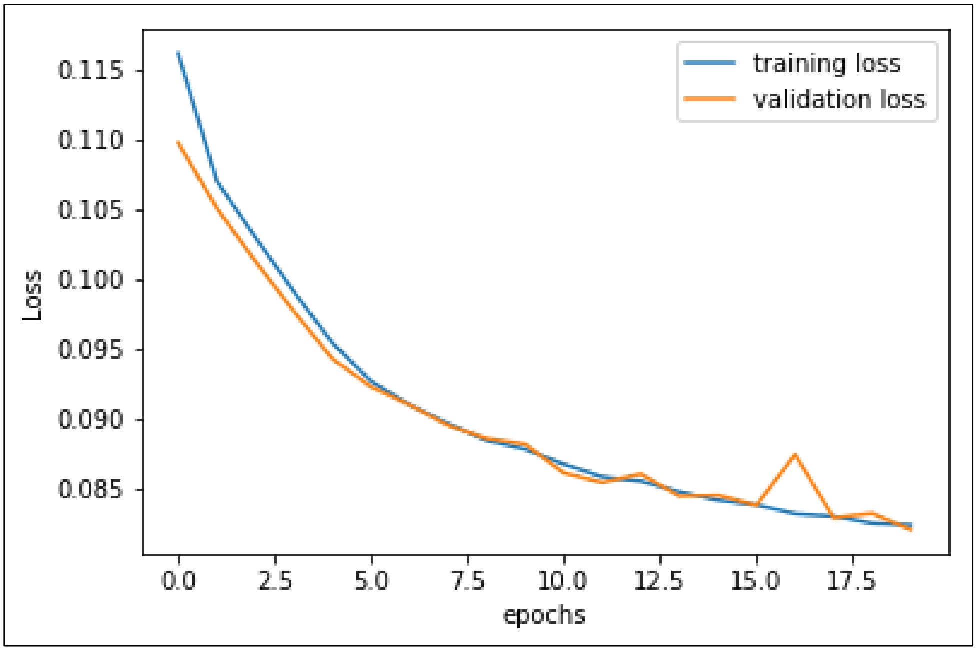

5.0
4.5
4.0
3.5
3.0
Loss
0
10
20
30
40
50
Epochs

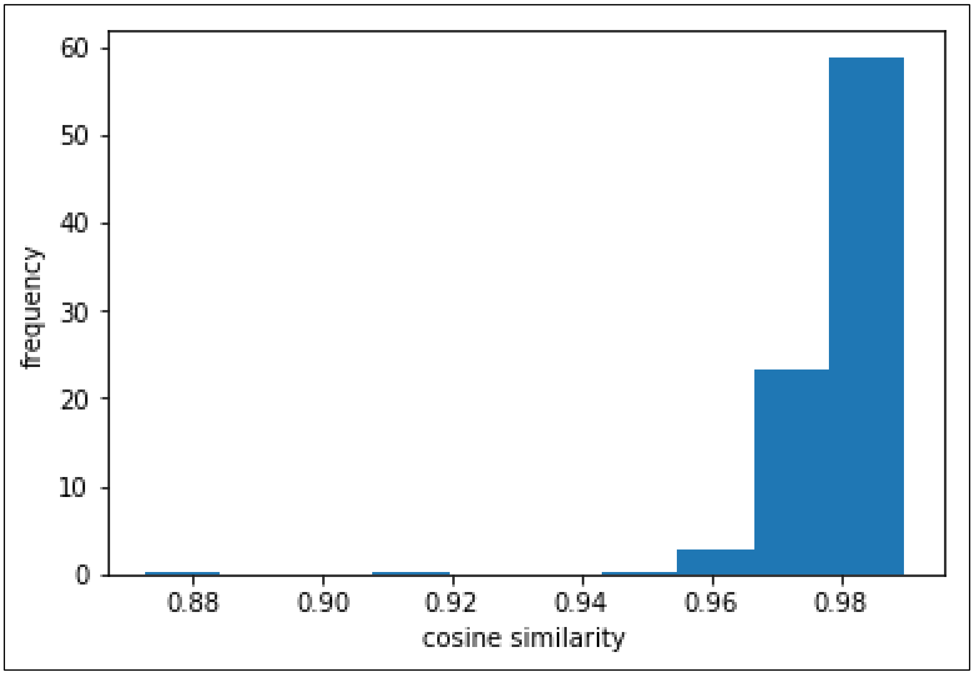



0.20
0.18
0.16
0.14
0.12
0.10
Loss
0
10
20
30
40
50
Epochs

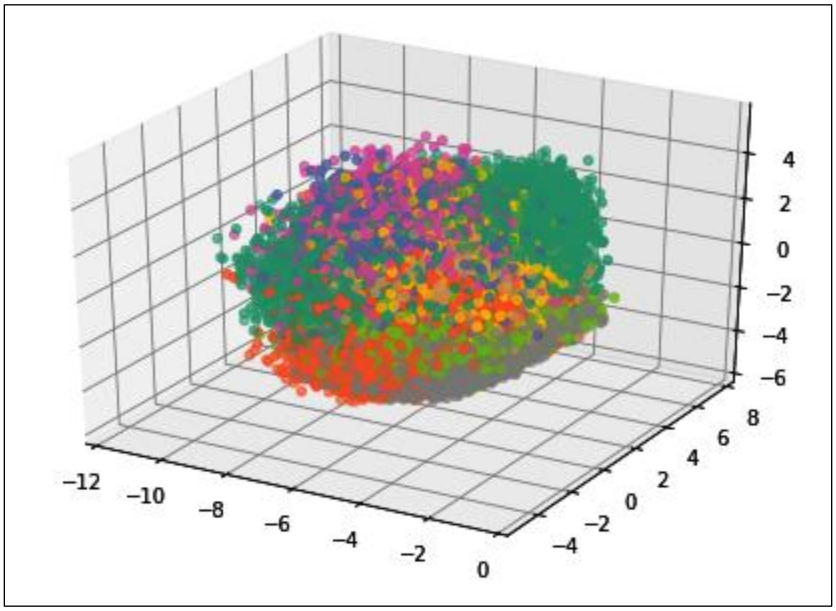


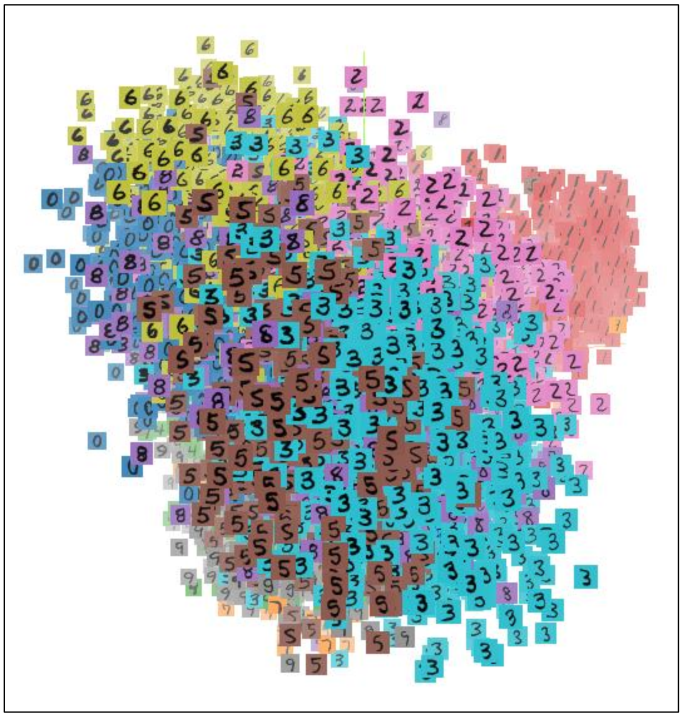

(None, SEQUENCE_LEN, EMBED_SIZE)
(None, 1024)
Encoder LSTM
Sentence
Vector
RepeatVector
(None, 1024, SEQUENCE_LEN)
Decoder LSTM
(None, SEQUENCE_LEN, EMBED_SIZE)

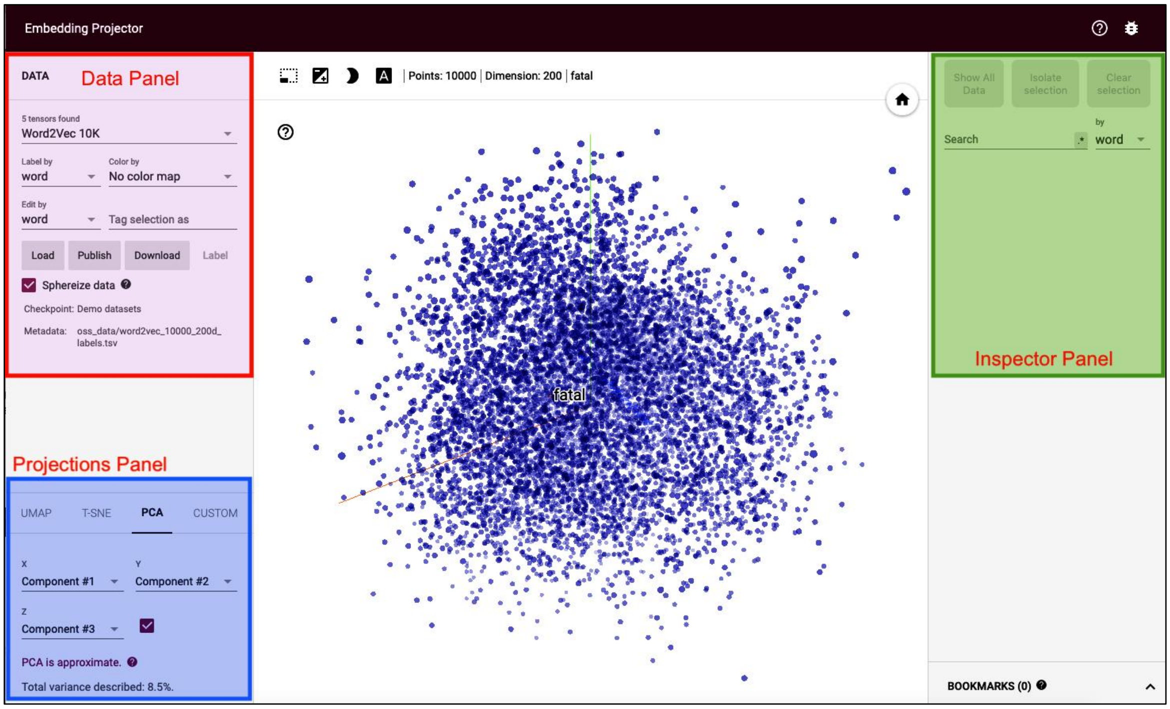

training loss
validation loss
Loss
0.115
0.110
0.105
0.100
0.095
0.090
0.085
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
epochs


frequency
60
50
40
30
20
10
0
0.88
0.90
0.92
0.94
0.96
0.98
cosine similarity

Chapter 10: Unsupervised Learning







5 tensors found: Word2Vec 10K

Label by: word

Color by: No color map

Edit by: word

Tag selection as

☑ Sphereize data

Checkpoint: Demo datasets

Metadata: oss_data/word2vec_10000_200d_labels.tsv

Points: 10000 | Dimension: 200 | fatal

X: Component #1

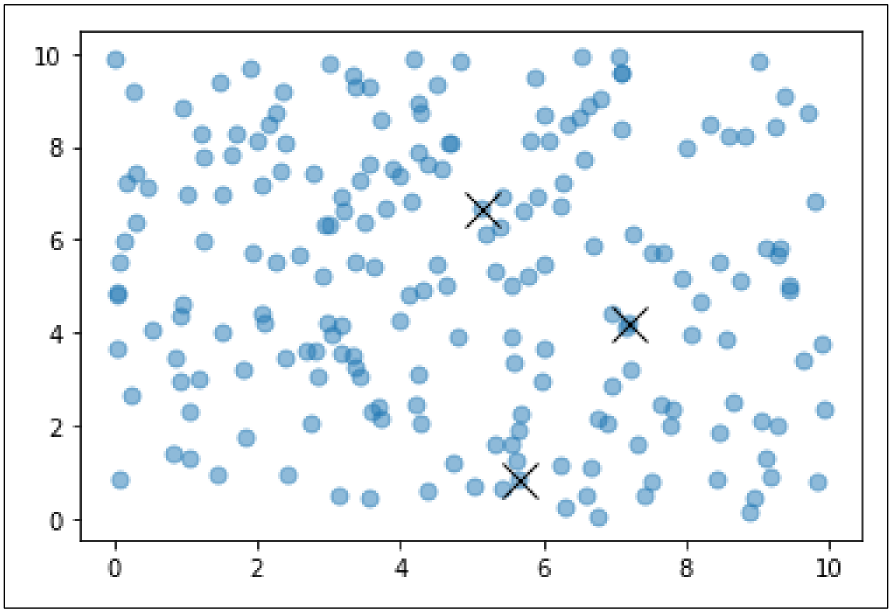
Y: Component #2

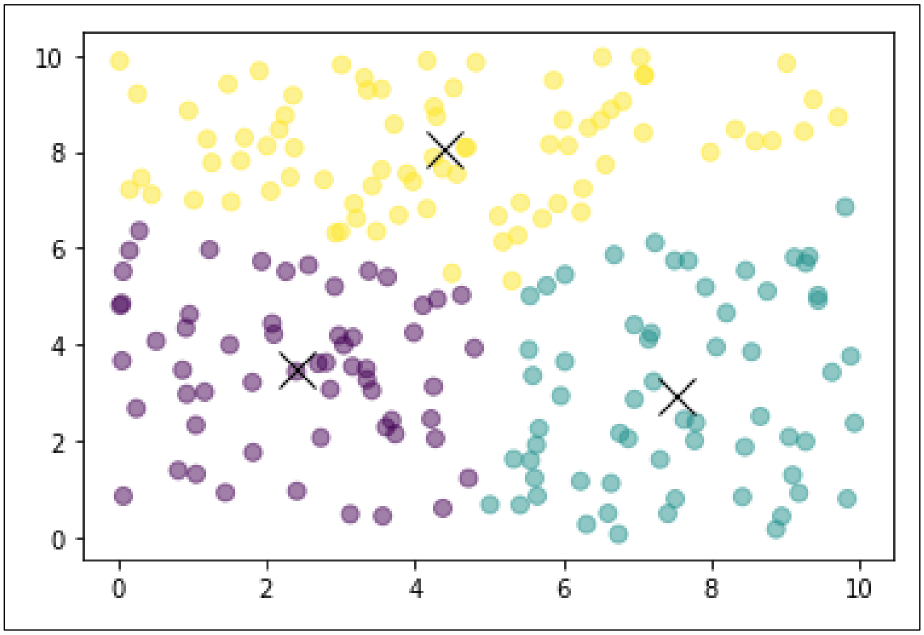
Z: Component #3 ☑

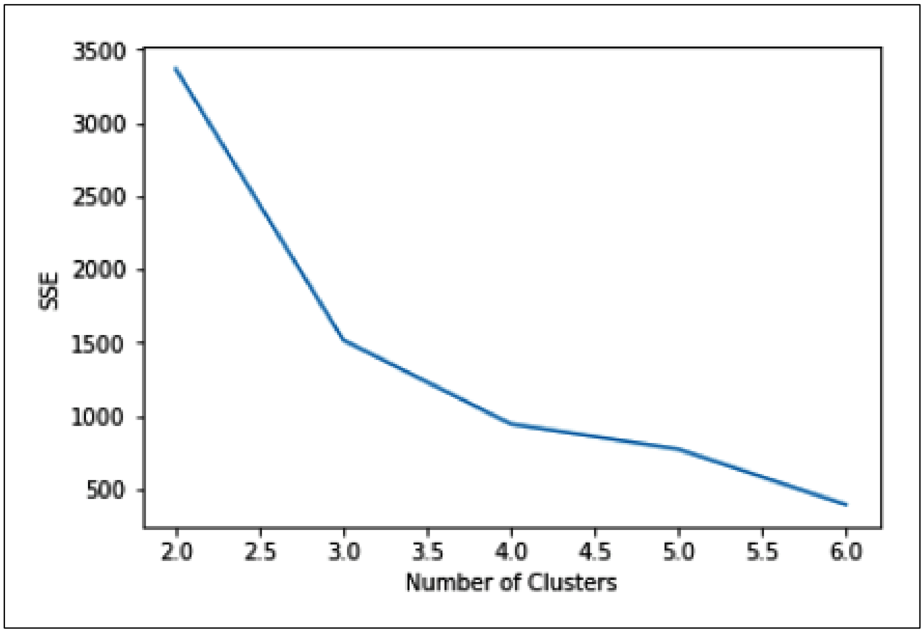
PCA is approximate.

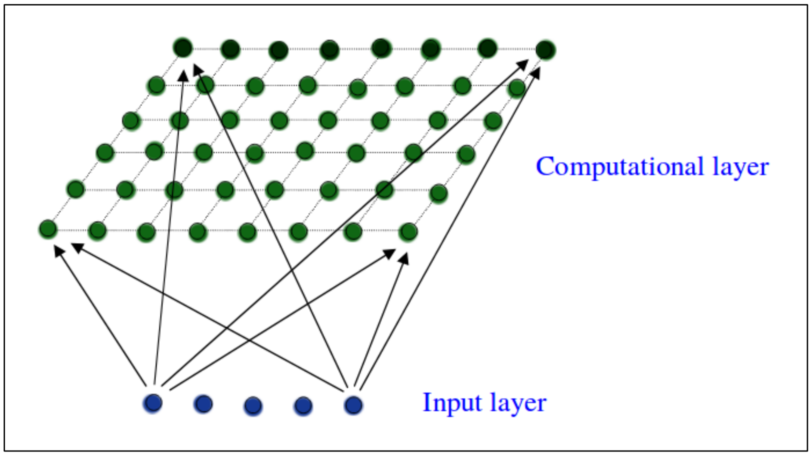
Total variance described: 8.5%.

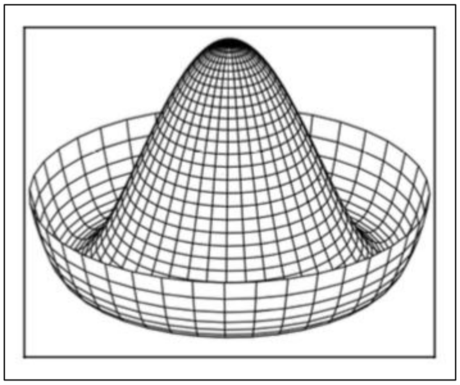
by: word

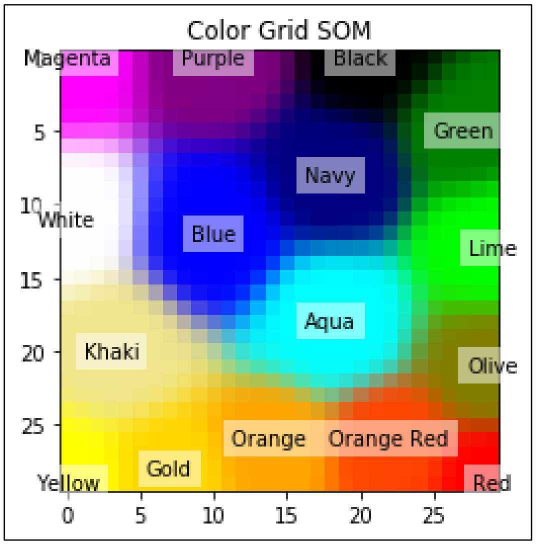


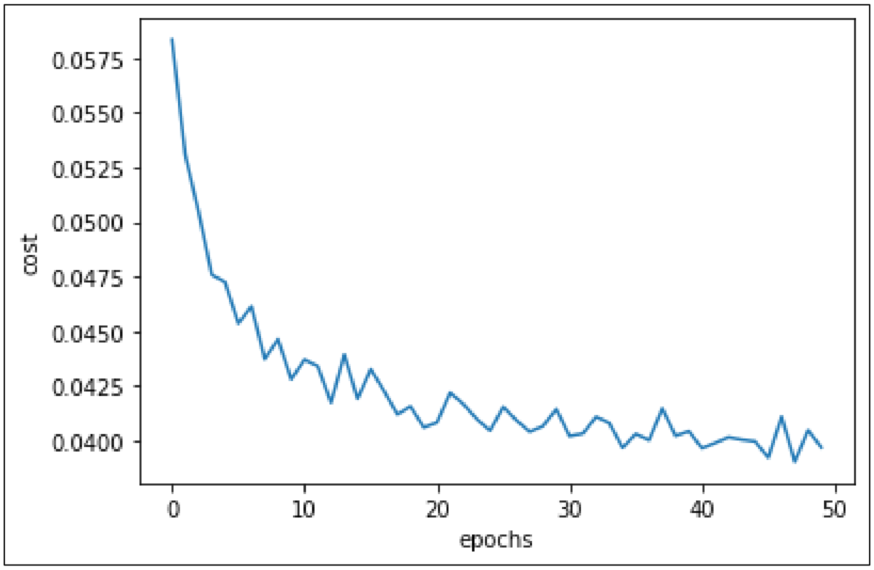

0
2
4
6
8
10
0
2
4
6
8
10

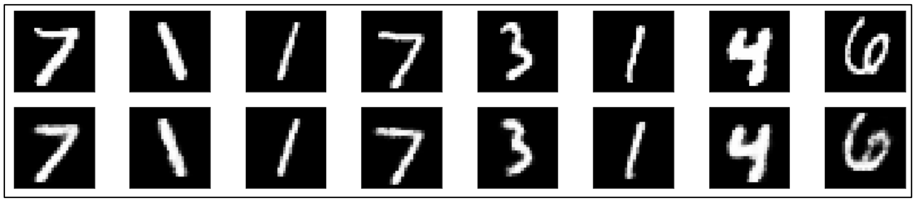

3500
3000
2500
2000
1500
1000
500
SSE
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
Number of Clusters

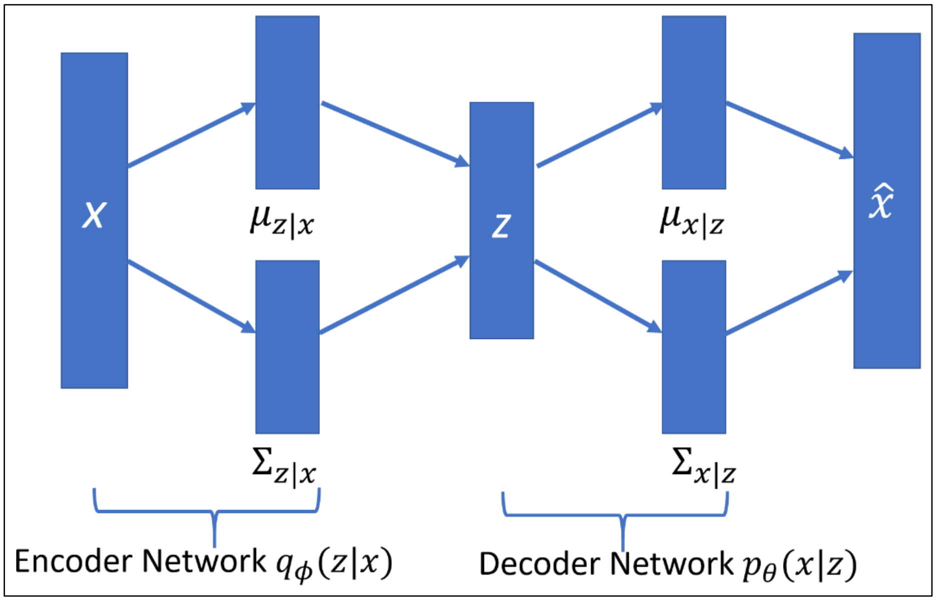

Computational layer
Input layer

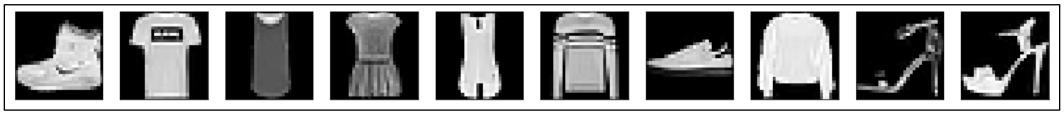


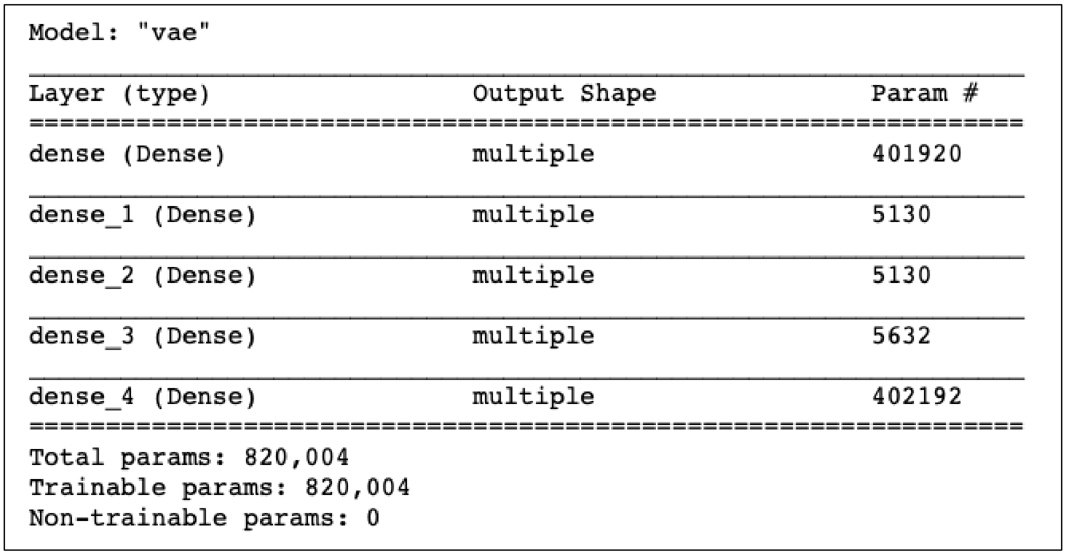

Color Grid SOM
Magenta
Purple
Black
Green
Navy
White
Blue
Lime
Aqua
Khaki
Olive
Orange
Orange Red
Gold
Yellow
Red
0
5
10
15
20
25
0
5
10
15
20
25


cost
0.0575
0.0550
0.0525
0.0500
0.0475
0.0450
0.0425
0.0400
0
10
20
30
40
50
epochs

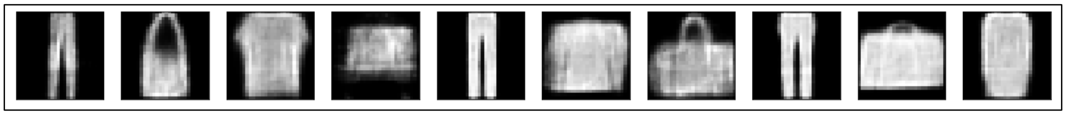



x
$\mu_{z|x}$
$\Sigma_{z|x}$
z
$\mu_{x|z}$
$\Sigma_{x|z}$
$\hat{x}$
Encoder Network $q_\phi(z|x)$
Decoder Network $p_\theta(x|z)$

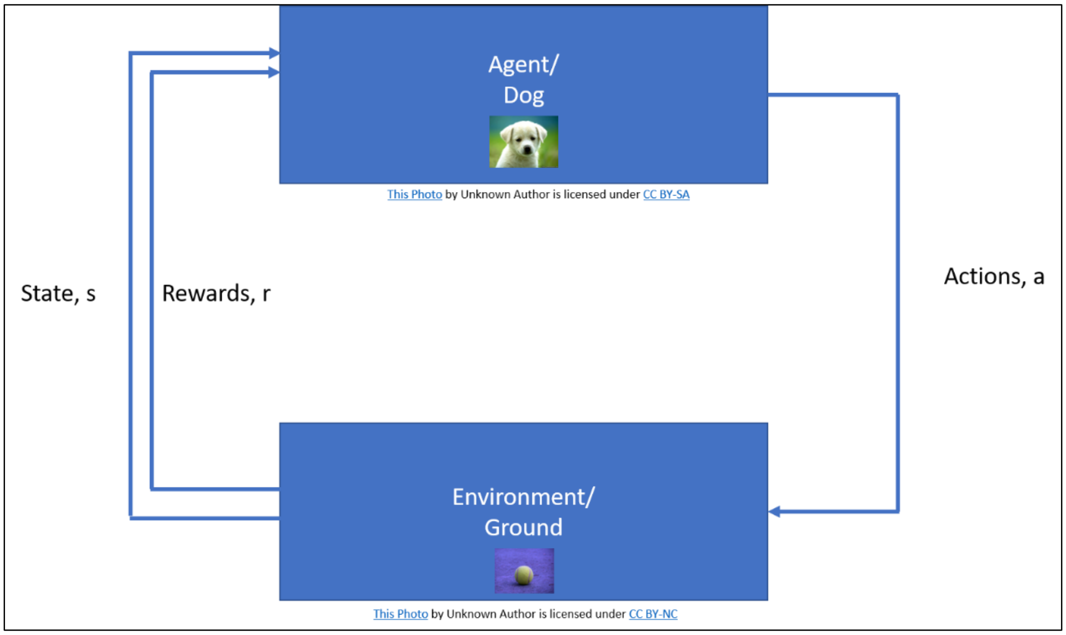




```
Model: "vae"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                multiple                  401920
_________________________________________________________________
dense_1 (Dense)              multiple                  5130
_________________________________________________________________
dense_2 (Dense)              multiple                  5130
_________________________________________________________________
dense_3 (Dense)              multiple                  5632
_________________________________________________________________
dense_4 (Dense)              multiple                  402192
=================================================================
Total params: 820,004
Trainable params: 820,004
Non-trainable params: 0
```



Chapter 11: Reinforcement Learning



Agent/
Dog

This Photo by Unknown Author is licensed under CC BY-SA

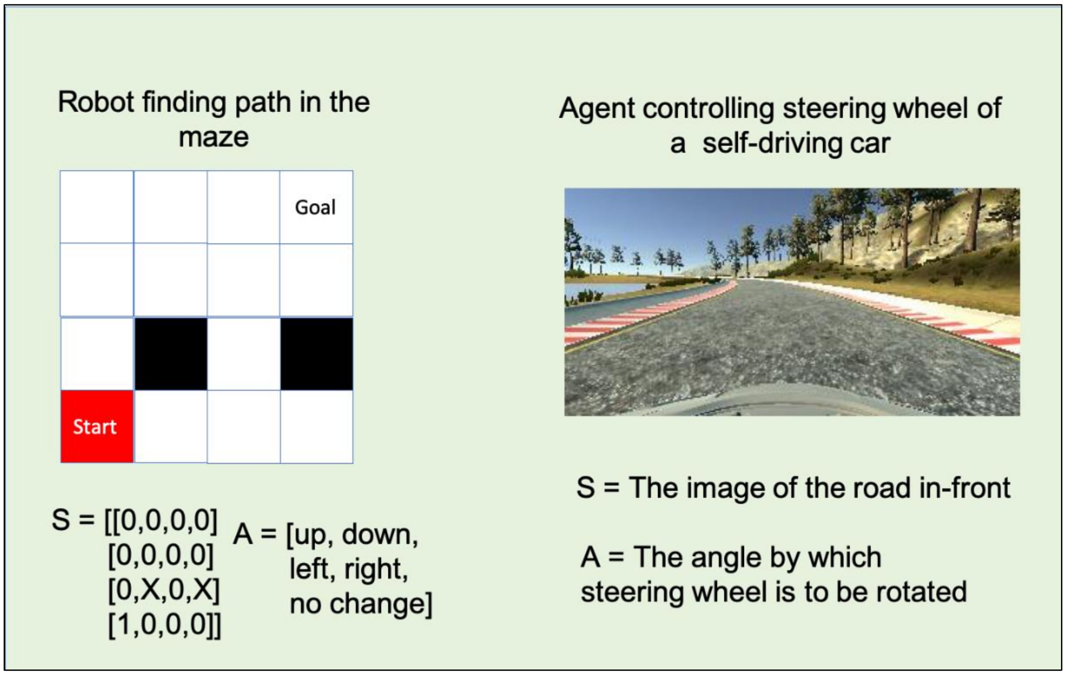
State, s

Rewards, r

Actions, a

Environment/
Ground

This Photo by Unknown Author is licensed under CC BY-NC



Robot finding path in the maze

			Goal
	■		■
Start			

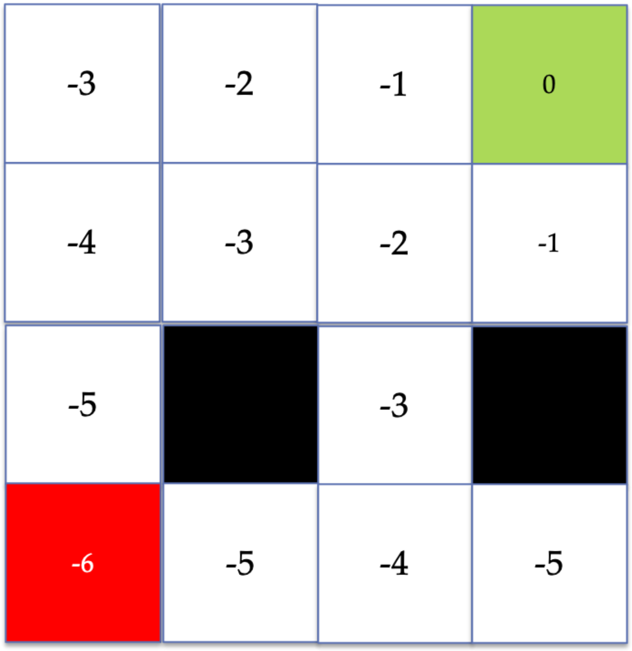
S = [[0,0,0,0]
[0,0,0,0]
[0,X,0,X]
[1,0,0,0]]

A = [up, down,
left, right,
no change]

Agent controlling steering wheel of a self-driving car

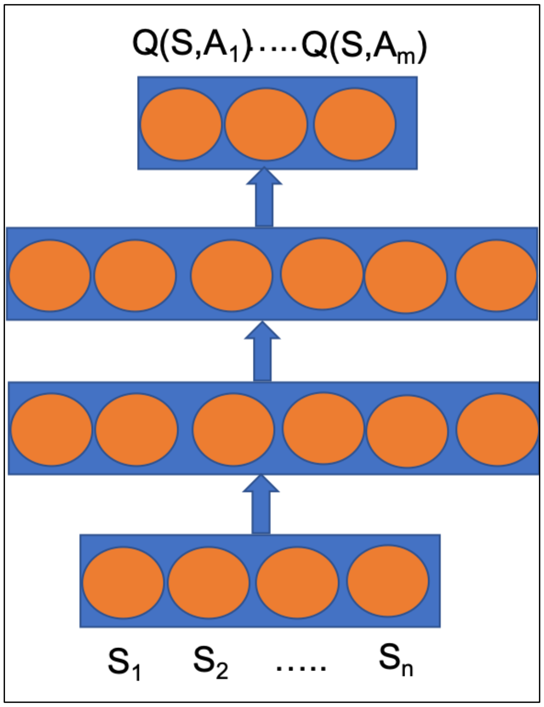
S = The image of the road in-front

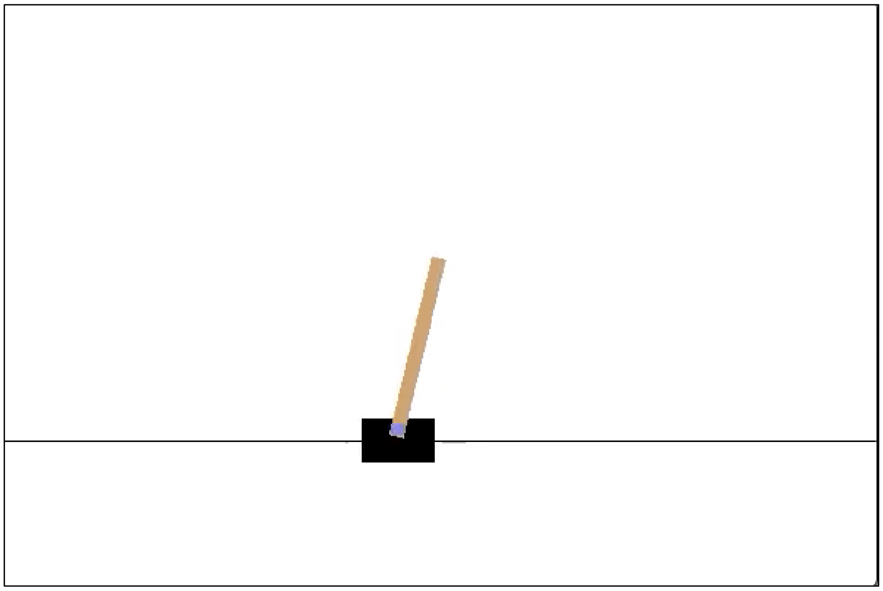
A = The angle by which steering wheel is to be rotated

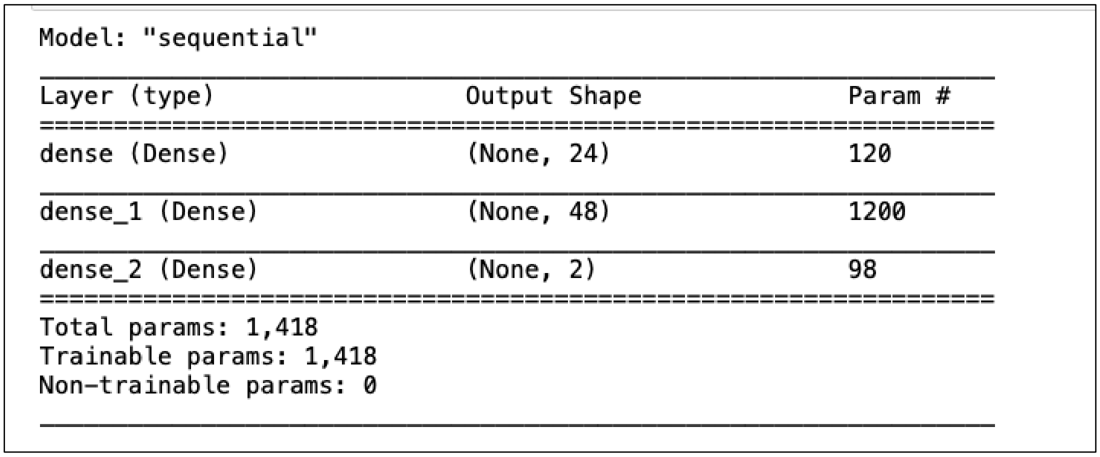


-3	-2	-1	0
-4	-3	-2	-1
-5		-3	
-6	-5	-4	-5

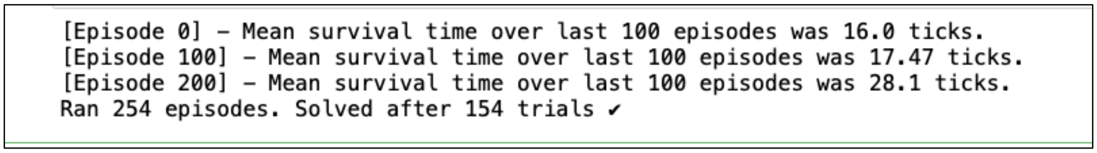
Each box has the value function:
Number of steps needed to reach goal (green box)


Q(S,A_1)….Q(S,A_m)
S_1
S_2
…..
S_n

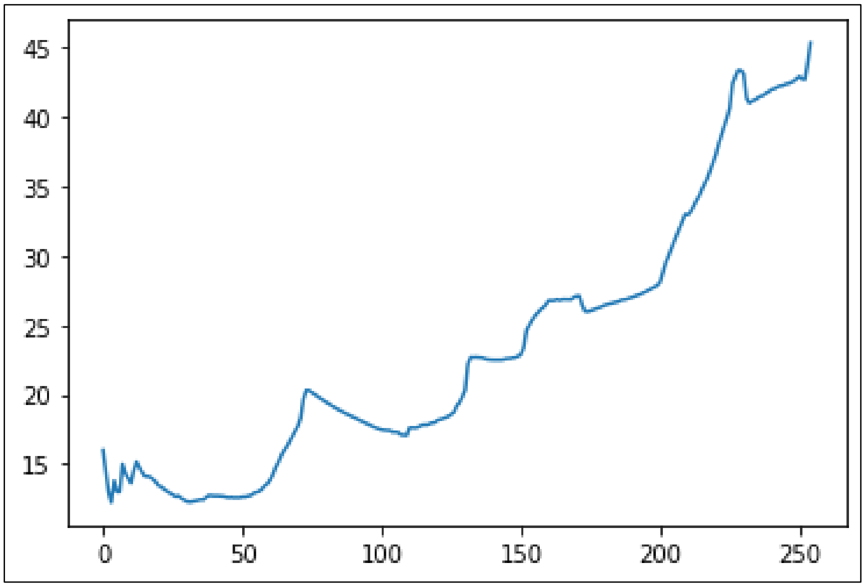




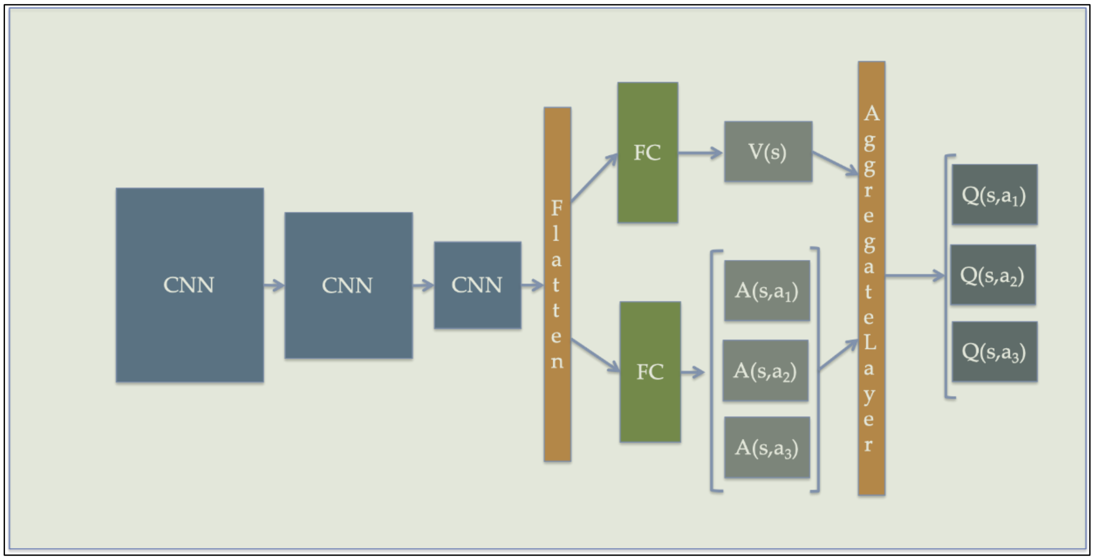
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 24)                120
_________________________________________________________________
dense_1 (Dense)              (None, 48)                1200
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 98
=================================================================
Total params: 1,418
Trainable params: 1,418
Non-trainable params: 0
_________________________________________________________________
```

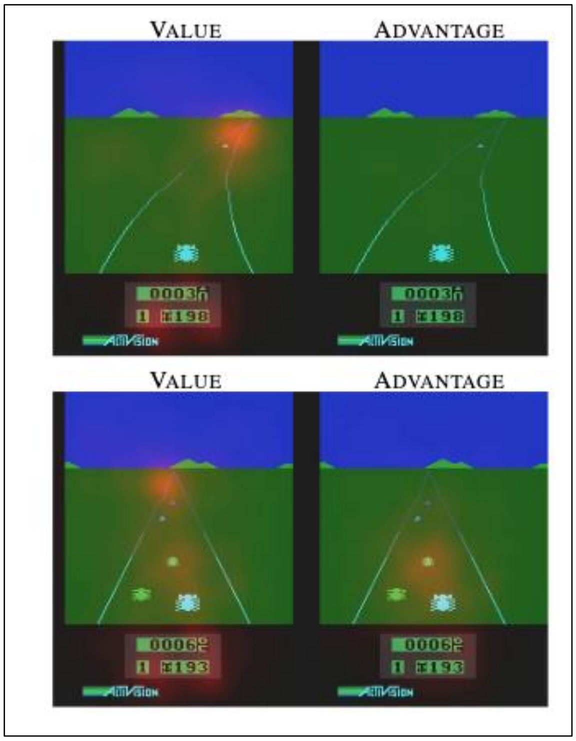


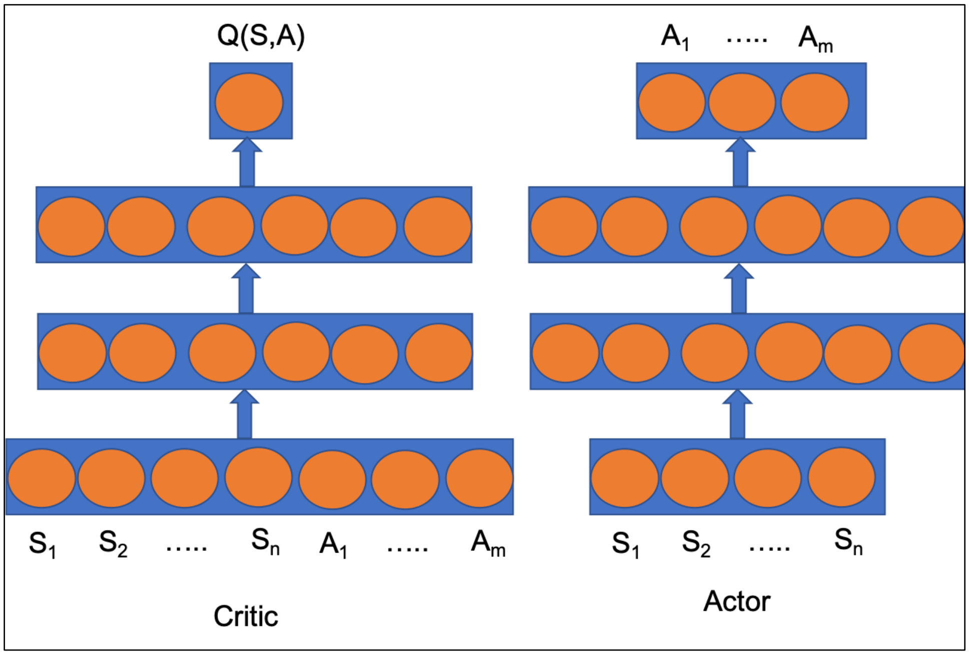
```
 [Episode 0] - Mean survival time over last 100 episodes was 16.0 ticks.
 [Episode 100] - Mean survival time over last 100 episodes was 17.47 ticks.
 [Episode 200] - Mean survival time over last 100 episodes was 28.1 ticks.
Ran 254 episodes. Solved after 154 trials ✔
```



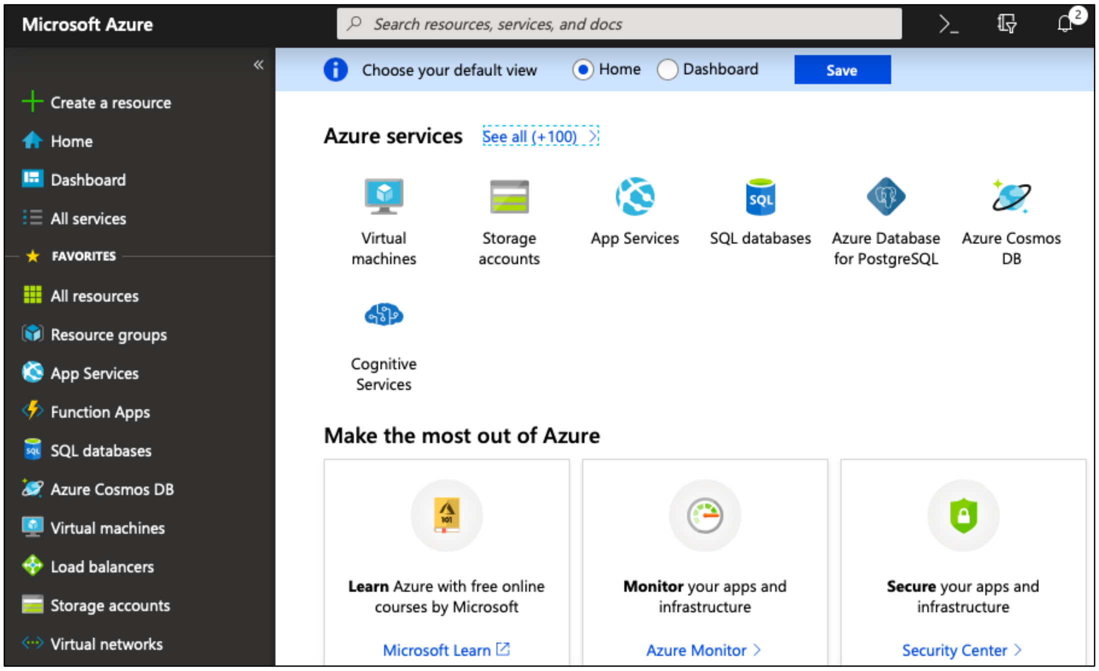
000 5 1
Episode 1


CNN
CNN
CNN
Flatten
FC
FC
V(s)
A(s,a$_1$)
A(s,a$_2$)
A(s,a$_3$)
AggregateLayer
Q(s,a$_1$)
Q(s,a$_2$)
Q(s,a$_3$)


VALUE
ADVANTAGE
VALUE
ADVANTAGE


Q(S,A)
A_1 ….. A_m
S_1 S_2 ….. S_n A_1 ….. A_m
Critic
S_1 S_2 ….. S_n
Actor

Chapter 12: TensorFlow and Cloud

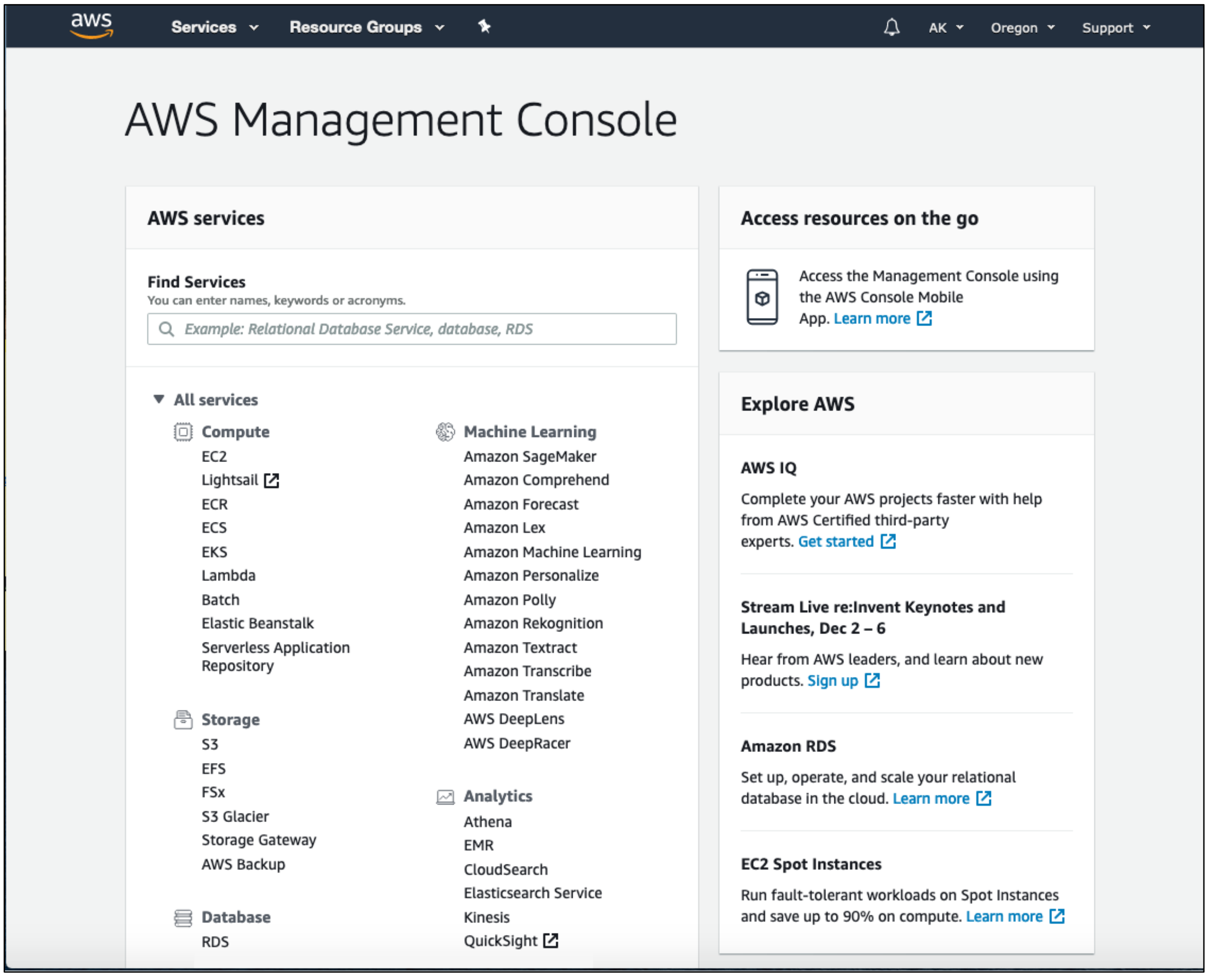


Choose your default view ● Home ○ Dashboard

Learn Azure with free online courses by Microsoft

Monitor your apps and infrastructure

Secure your apps and infrastructure



AWS Management Console

AWS services

Find Services
You can enter names, keywords or acronyms.

Example: Relational Database Service, database, RDS

▼ **All services**

Compute
- EC2
- Lightsail
- ECR
- ECS
- EKS
- Lambda
- Batch
- Elastic Beanstalk
- Serverless Application Repository

Storage
- S3
- EFS
- FSx
- S3 Glacier
- Storage Gateway
- AWS Backup

Database
- RDS

Machine Learning
- Amazon SageMaker
- Amazon Comprehend
- Amazon Forecast
- Amazon Lex
- Amazon Machine Learning
- Amazon Personalize
- Amazon Polly
- Amazon Rekognition
- Amazon Textract
- Amazon Transcribe
- Amazon Translate
- AWS DeepLens
- AWS DeepRacer

Analytics
- Athena
- EMR
- CloudSearch
- Elasticsearch Service
- Kinesis
- QuickSight

Access resources on the go

Access the Management Console using the AWS Console Mobile App. Learn more

Explore AWS

AWS IQ

Complete your AWS projects faster with help from AWS Certified third-party experts. Get started

Stream Live re:Invent Keynotes and Launches, Dec 2 – 6

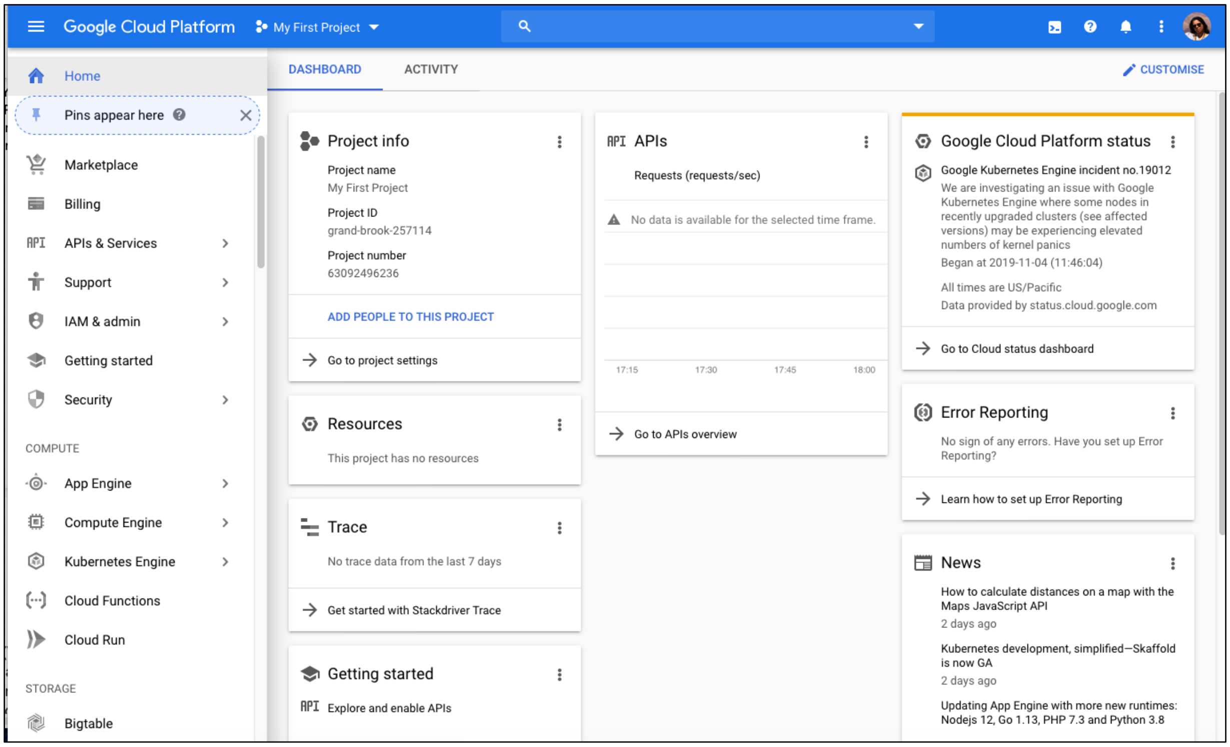
Hear from AWS leaders, and learn about new products. Sign up

Amazon RDS

Set up, operate, and scale your relational database in the cloud. Learn more

EC2 Spot Instances

Run fault-tolerant workloads on Spot Instances and save up to 90% on compute. Learn more



Project info

Project name
My First Project

Project ID
grand-brook-257114

Project number
63092496236

ADD PEOPLE TO THIS PROJECT

Go to project settings

Resources

This project has no resources

Trace

No trace data from the last 7 days

Get started with Stackdriver Trace

Getting started

Explore and enable APIs

APIs

Requests (requests/sec)

No data is available for the selected time frame.

Go to APIs overview

Google Cloud Platform status

Google Kubernetes Engine incident no.19012
We are investigating an issue with Google Kubernetes Engine where some nodes in recently upgraded clusters (see affected versions) may be experiencing elevated numbers of kernel panics
Began at 2019-11-04 (11:46:04)

All times are US/Pacific
Data provided by status.cloud.google.com

Go to Cloud status dashboard

Error Reporting

No sign of any errors. Have you set up Error Reporting?

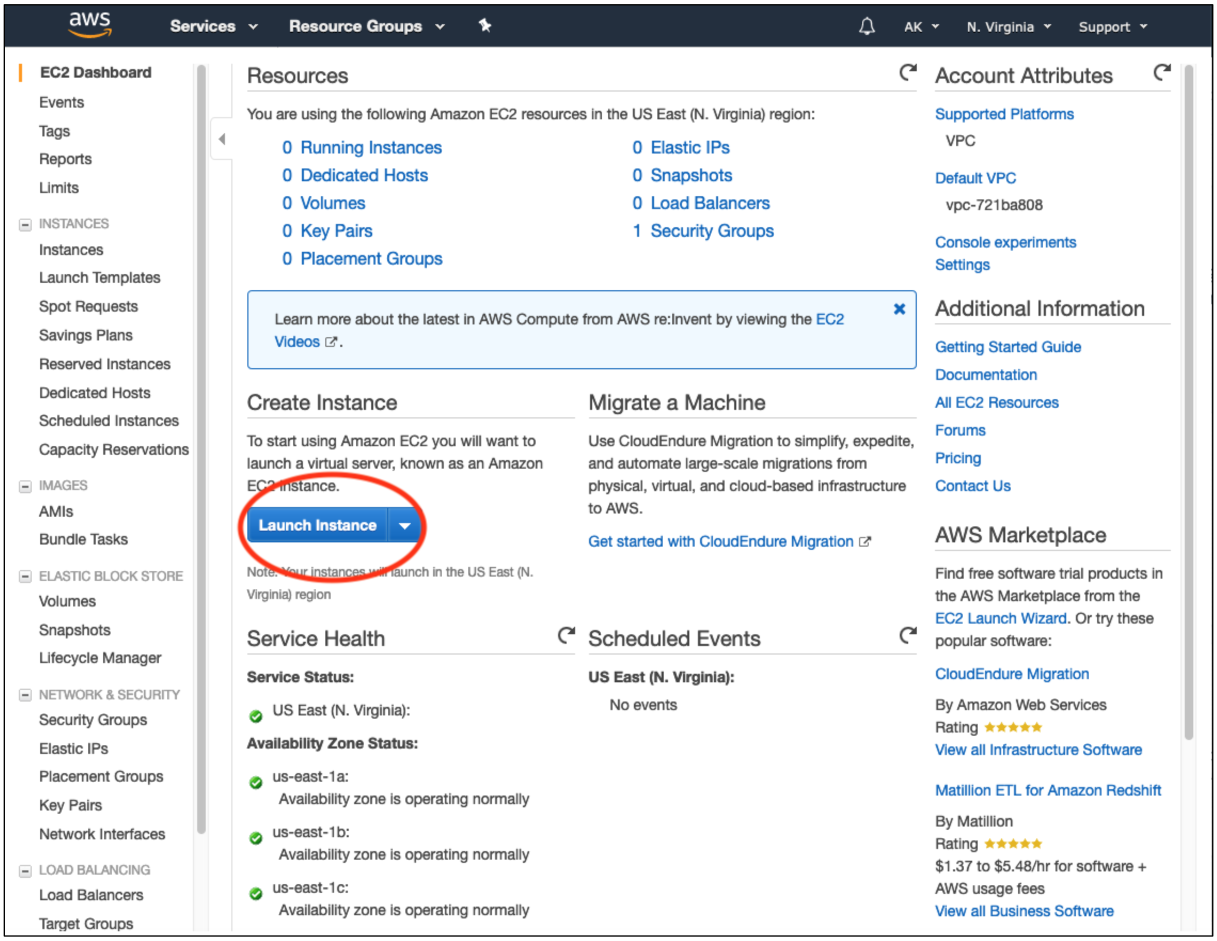
Learn how to set up Error Reporting

News

How to calculate distances on a map with the Maps JavaScript API
2 days ago

Kubernetes development, simplified—Skaffold is now GA
2 days ago

Updating App Engine with more new runtimes: Nodejs 12, Go 1.13, PHP 7.3 and Python 3.8



Resources

You are using the following Amazon EC2 resources in the US East (N. Virginia) region:

0 Running Instances	0 Elastic IPs
0 Dedicated Hosts	0 Snapshots
0 Volumes	0 Load Balancers
0 Key Pairs	1 Security Groups
0 Placement Groups	

Learn more about the latest in AWS Compute from AWS re:Invent by viewing the EC2 Videos.

Create Instance

To start using Amazon EC2 you will want to launch a virtual server, known as an Amazon EC2 instance.

Note: Your instances will launch in the US East (N. Virginia) region

Migrate a Machine

Use CloudEndure Migration to simplify, expedite, and automate large-scale migrations from physical, virtual, and cloud-based infrastructure to AWS.

Get started with CloudEndure Migration

Service Health

Service Status:

✓ US East (N. Virginia):

Availability Zone Status:

✓ us-east-1a:
Availability zone is operating normally

✓ us-east-1b:
Availability zone is operating normally

✓ us-east-1c:
Availability zone is operating normally

Scheduled Events

US East (N. Virginia):

No events

Account Attributes

Supported Platforms
VPC

Default VPC
vpc-721ba808

Console experiments
Settings

Additional Information

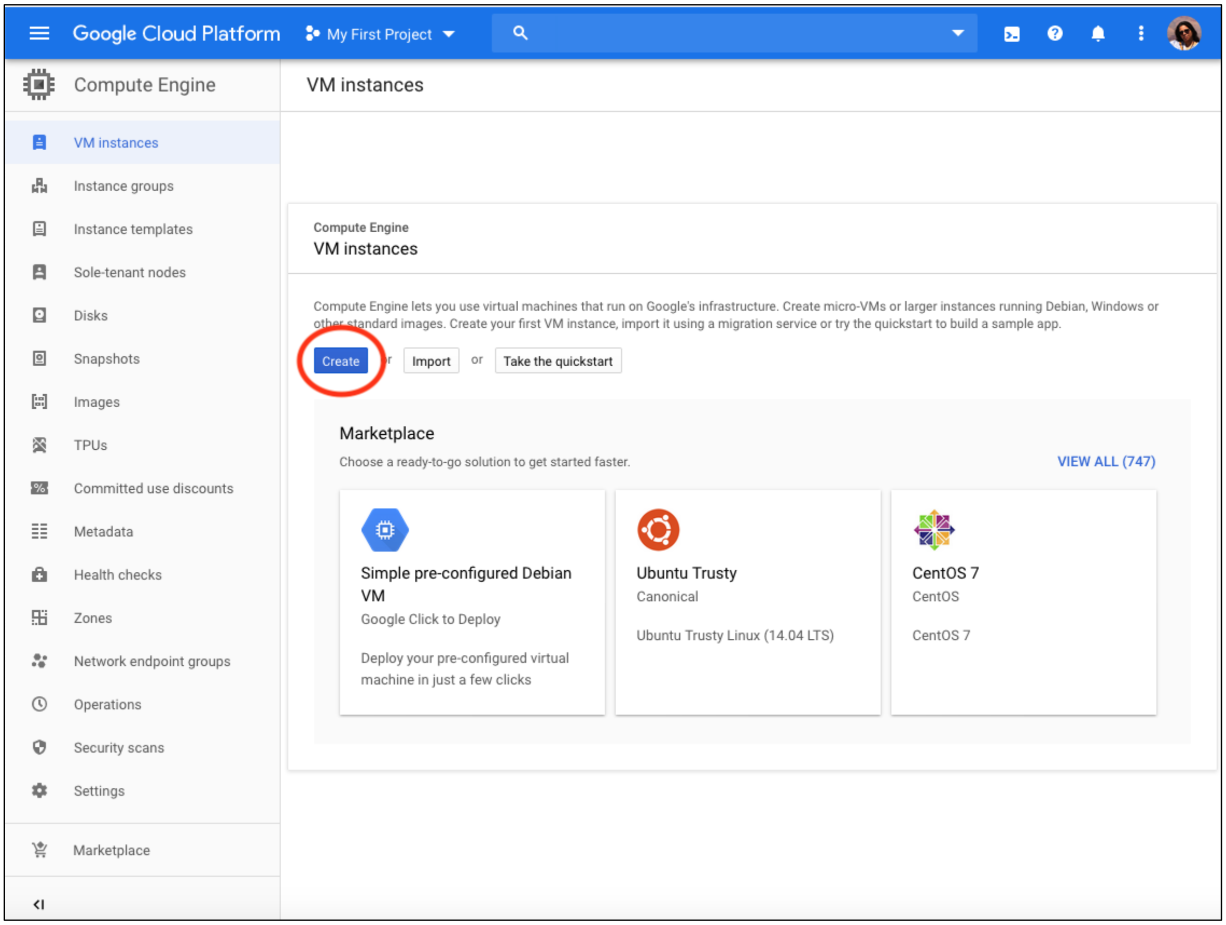
Getting Started Guide
Documentation
All EC2 Resources
Forums
Pricing
Contact Us

AWS Marketplace

Find free software trial products in the AWS Marketplace from the EC2 Launch Wizard. Or try these popular software:

CloudEndure Migration
By Amazon Web Services
Rating ★★★★★
View all Infrastructure Software

Matillion ETL for Amazon Redshift
By Matillion
Rating ★★★★★
$1.37 to $5.48/hr for software + AWS usage fees
View all Business Software



Compute Engine

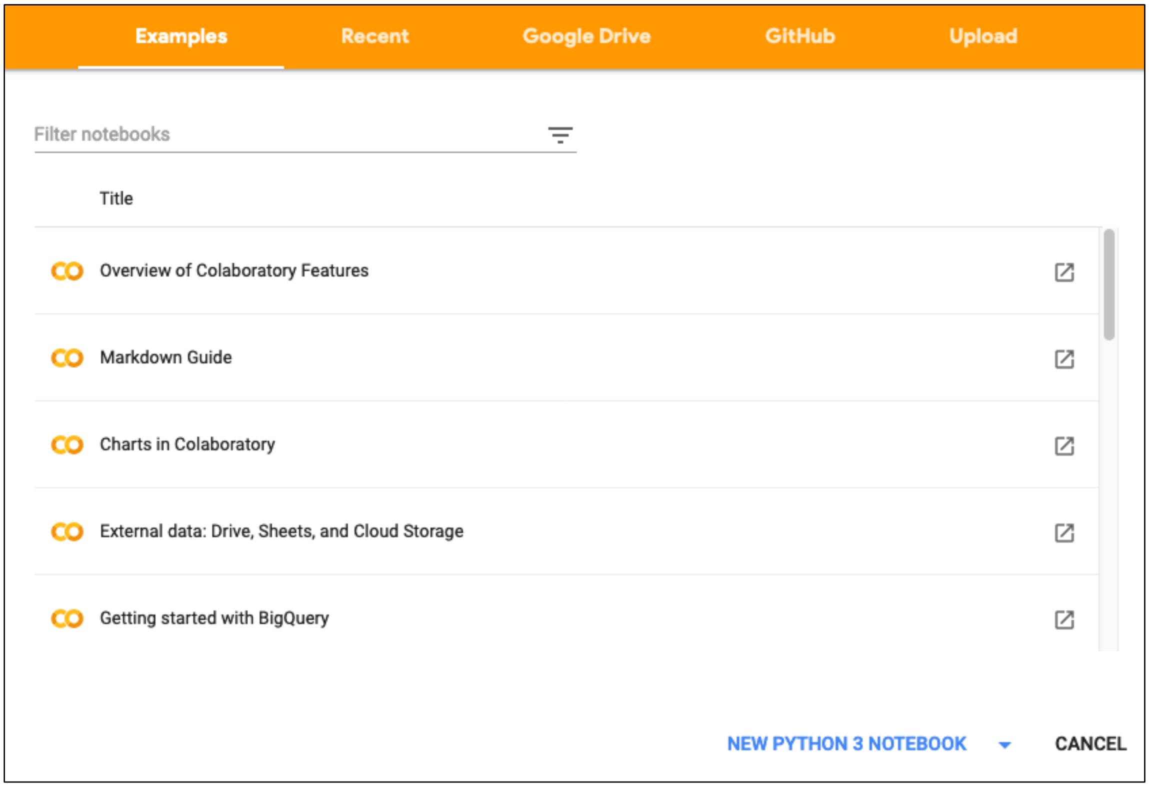
VM instances

Compute Engine lets you use virtual machines that run on Google's infrastructure. Create micro-VMs or larger instances running Debian, Windows or other standard images. Create your first VM instance, import it using a migration service or try the quickstart to build a sample app.

Marketplace

Choose a ready-to-go solution to get started faster.

Simple pre-configured Debian VM	Ubuntu Trusty	CentOS 7
Google Click to Deploy	Canonical	CentOS
Deploy your pre-configured virtual machine in just a few clicks	Ubuntu Trusty Linux (14.04 LTS)	CentOS 7



Title

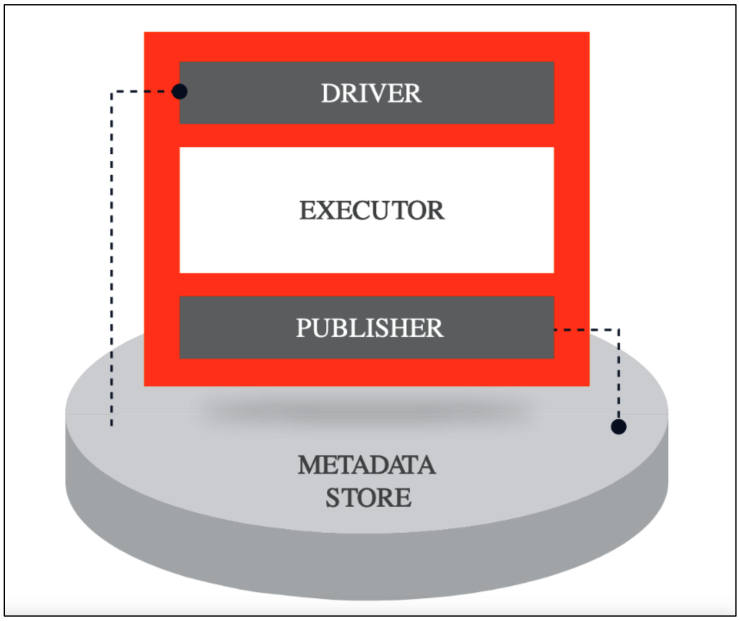
Overview of Colaboratory Features

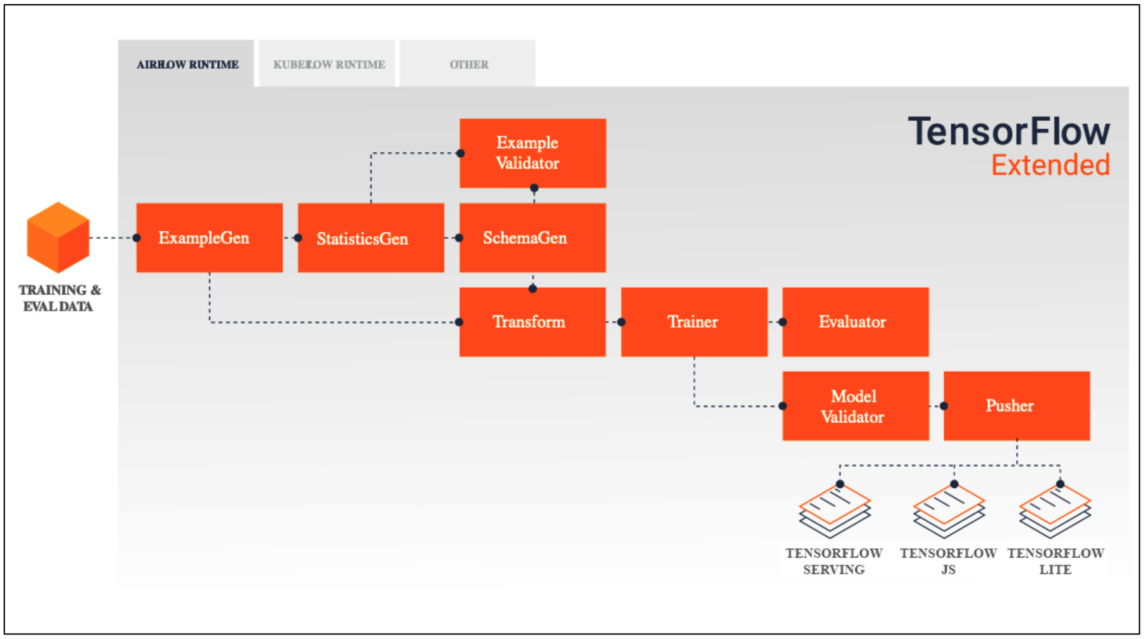
Markdown Guide

Charts in Colaboratory

External data: Drive, Sheets, and Cloud Storage

Getting started with BigQuery





AIRFLOW RUNTIME

KUBEFLOW RUNTIME

OTHER

TensorFlow Extended

TRAINING & EVAL DATA

ExampleGen

StatisticsGen

SchemaGen

Example Validator

Transform

Trainer

Evaluator

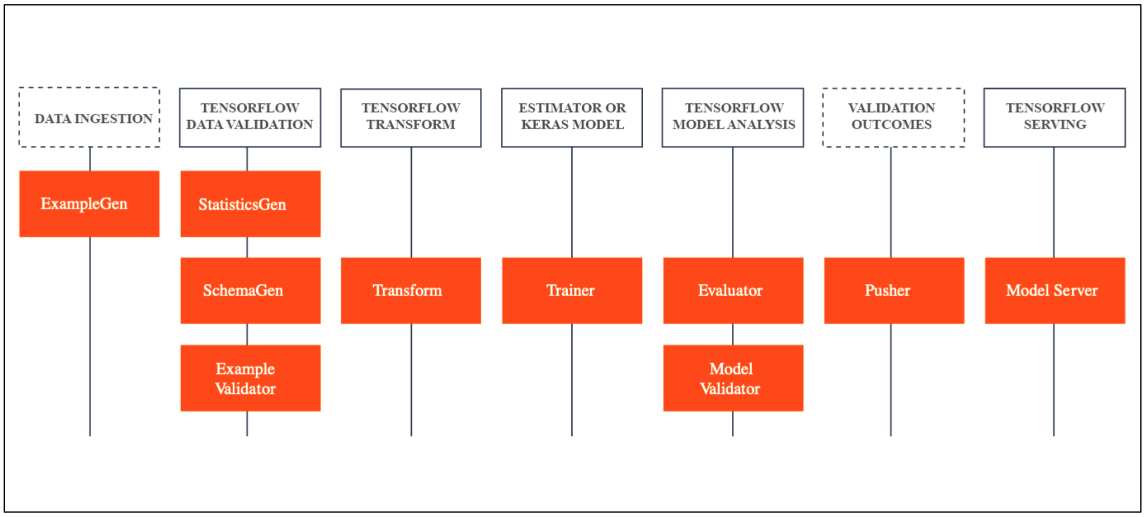
Model Validator

Pusher

TENSORFLOW SERVING

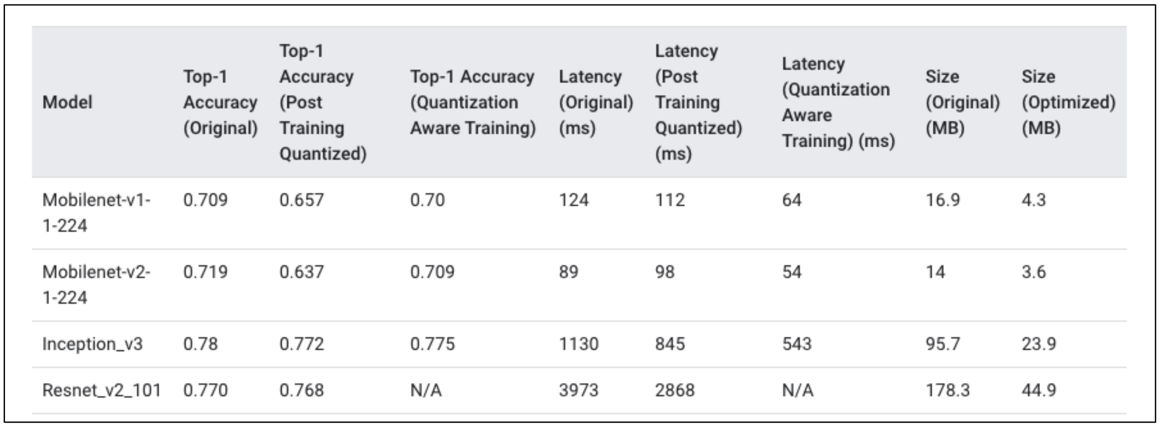
TENSORFLOW JS

TENSORFLOW LITE

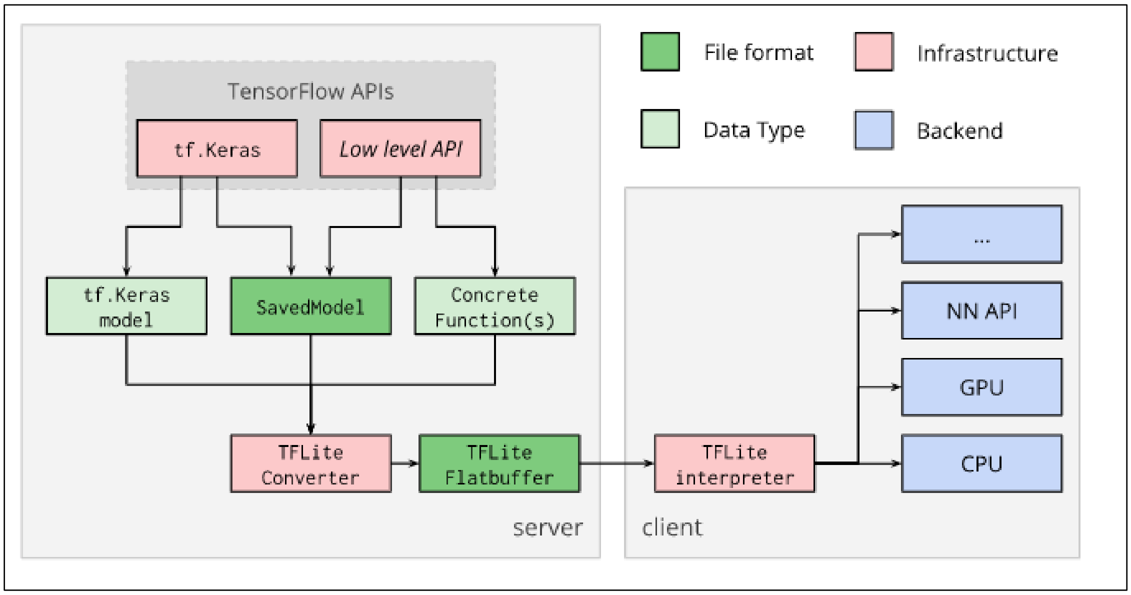


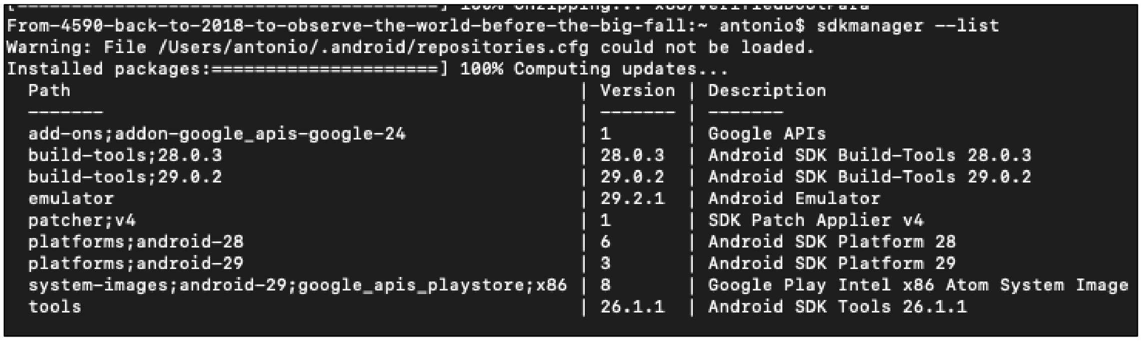
DATA INGESTION	TENSORFLOW DATA VALIDATION	TENSORFLOW TRANSFORM	ESTIMATOR OR KERAS MODEL	TENSORFLOW MODEL ANALYSIS	VALIDATION OUTCOMES	TENSORFLOW SERVING
ExampleGen	StatisticsGen					
	SchemaGen	Transform	Trainer	Evaluator	Pusher	Model Server
	Example Validator			Model Validator		

Chapter 13: TensorFlow for Mobile and IoT and TensorFlow.js

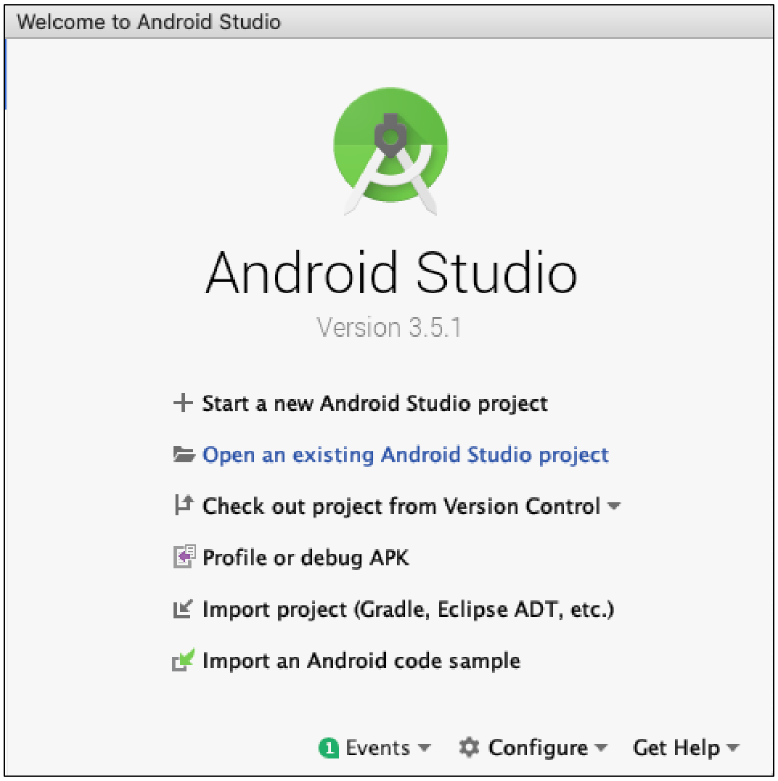


Model	Top-1 Accuracy (Original)	Top-1 Accuracy (Post Training Quantized)	Top-1 Accuracy (Quantization Aware Training)	Latency (Original) (ms)	Latency (Post Training Quantized) (ms)	Latency (Quantization Aware Training) (ms)	Size (Original) (MB)	Size (Optimized) (MB)
Mobilenet-v1-1-224	0.709	0.657	0.70	124	112	64	16.9	4.3
Mobilenet-v2-1-224	0.719	0.637	0.709	89	98	54	14	3.6
Inception_v3	0.78	0.772	0.775	1130	845	543	95.7	23.9
Resnet_v2_101	0.770	0.768	N/A	3973	2868	N/A	178.3	44.9

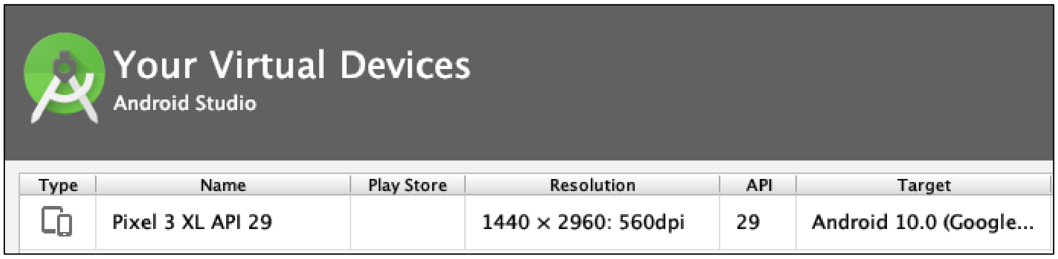




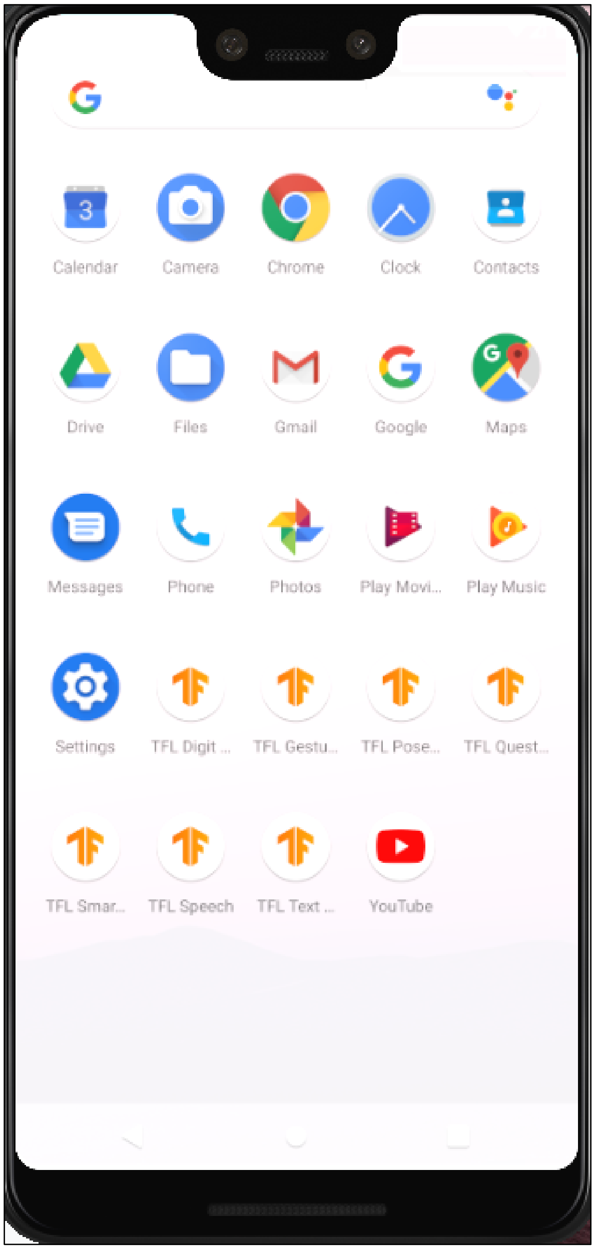
```
[=======================================] 100% Unzipping... x86/VerifiedBootPara
From-4590-back-to-2018-to-observe-the-world-before-the-big-fall:~ antonio$ sdkmanager --list
Warning: File /Users/antonio/.android/repositories.cfg could not be loaded.
Installed packages:=====================] 100% Computing updates...
  Path                                              | Version | Description
  -------                                           | ------- | -------
  add-ons;addon-google_apis-google-24               | 1       | Google APIs
  build-tools;28.0.3                                | 28.0.3  | Android SDK Build-Tools 28.0.3
  build-tools;29.0.2                                | 29.0.2  | Android SDK Build-Tools 29.0.2
  emulator                                          | 29.2.1  | Android Emulator
  patcher;v4                                        | 1       | SDK Patch Applier v4
  platforms;android-28                              | 6       | Android SDK Platform 28
  platforms;android-29                              | 3       | Android SDK Platform 29
  system-images;android-29;google_apis_playstore;x86 | 8       | Google Play Intel x86 Atom System Image
  tools                                             | 26.1.1  | Android SDK Tools 26.1.1
```

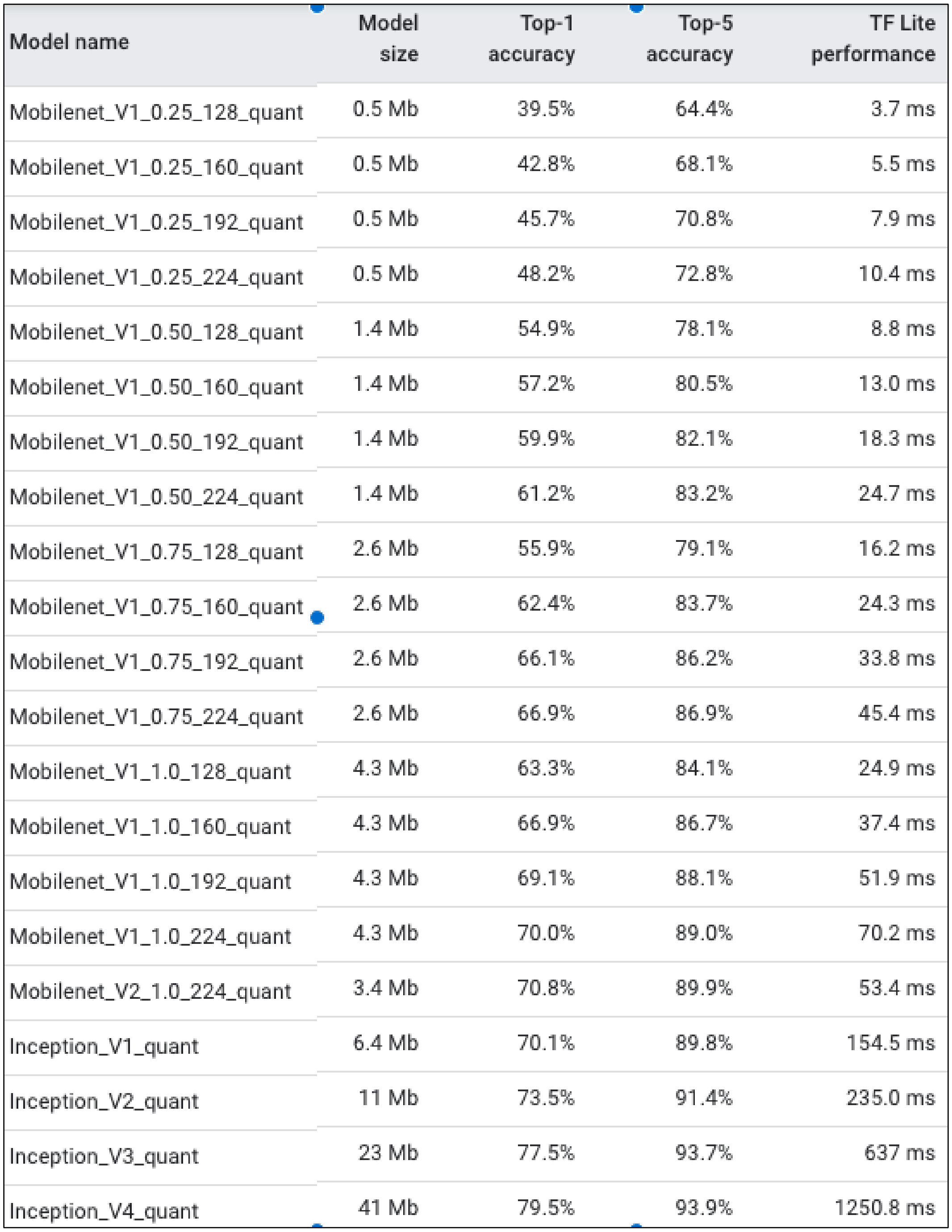


Welcome to Android Studio
Android Studio
Version 3.5.1
+ Start a new Android Studio project
Open an existing Android Studio project
Check out project from Version Control
Profile or debug APK
Import project (Gradle, Eclipse ADT, etc.)
Import an Android code sample
1 Events
Configure
Get Help

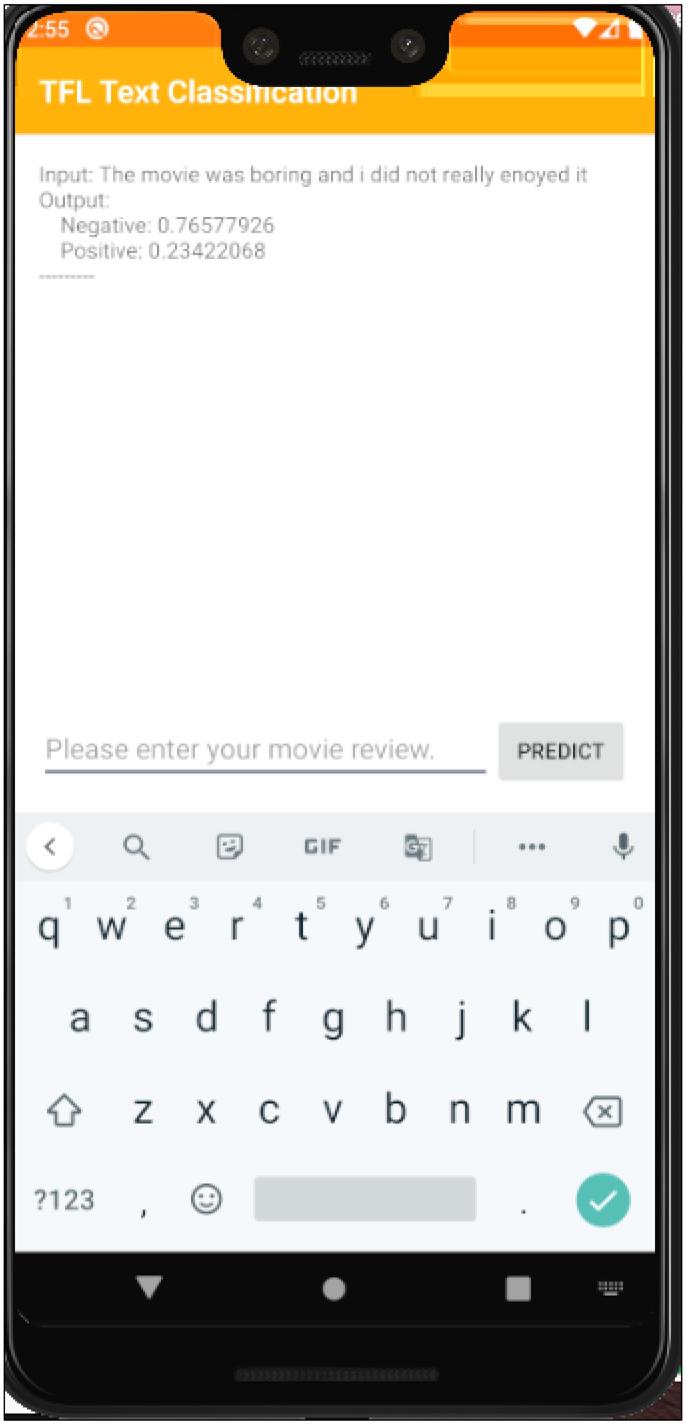


Type	Name	Play Store	Resolution	API	Target
	Pixel 3 XL API 29		1440 × 2960: 560dpi	29	Android 10.0 (Google...


Calendar
Camera
Chrome
Clock
Contacts
Drive
Files
Gmail
Google
Maps
Messages
Phone
Photos
Play Movi...
Play Music
Settings
TFL Digit ...
TFL Gestu...
TFL Pose...
TFL Quest...
TFL Smar...
TFL Speech
TFL Text ...
YouTube

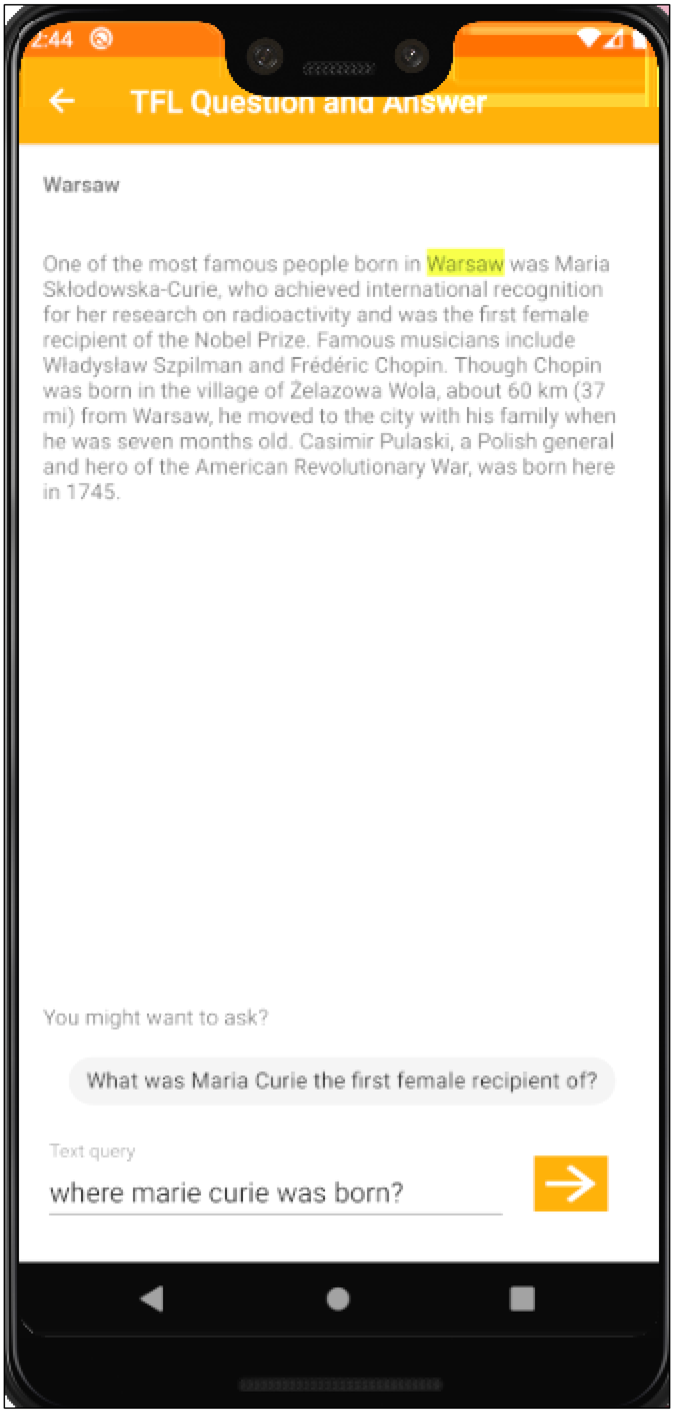


Model name	Model size	Top-1 accuracy	Top-5 accuracy	TF Lite performance
Mobilenet_V1_0.25_128_quant	0.5 Mb	39.5%	64.4%	3.7 ms
Mobilenet_V1_0.25_160_quant	0.5 Mb	42.8%	68.1%	5.5 ms
Mobilenet_V1_0.25_192_quant	0.5 Mb	45.7%	70.8%	7.9 ms
Mobilenet_V1_0.25_224_quant	0.5 Mb	48.2%	72.8%	10.4 ms
Mobilenet_V1_0.50_128_quant	1.4 Mb	54.9%	78.1%	8.8 ms
Mobilenet_V1_0.50_160_quant	1.4 Mb	57.2%	80.5%	13.0 ms
Mobilenet_V1_0.50_192_quant	1.4 Mb	59.9%	82.1%	18.3 ms
Mobilenet_V1_0.50_224_quant	1.4 Mb	61.2%	83.2%	24.7 ms
Mobilenet_V1_0.75_128_quant	2.6 Mb	55.9%	79.1%	16.2 ms
Mobilenet_V1_0.75_160_quant	2.6 Mb	62.4%	83.7%	24.3 ms
Mobilenet_V1_0.75_192_quant	2.6 Mb	66.1%	86.2%	33.8 ms
Mobilenet_V1_0.75_224_quant	2.6 Mb	66.9%	86.9%	45.4 ms
Mobilenet_V1_1.0_128_quant	4.3 Mb	63.3%	84.1%	24.9 ms
Mobilenet_V1_1.0_160_quant	4.3 Mb	66.9%	86.7%	37.4 ms
Mobilenet_V1_1.0_192_quant	4.3 Mb	69.1%	88.1%	51.9 ms
Mobilenet_V1_1.0_224_quant	4.3 Mb	70.0%	89.0%	70.2 ms
Mobilenet_V2_1.0_224_quant	3.4 Mb	70.8%	89.9%	53.4 ms
Inception_V1_quant	6.4 Mb	70.1%	89.8%	154.5 ms
Inception_V2_quant	11 Mb	73.5%	91.4%	235.0 ms
Inception_V3_quant	23 Mb	77.5%	93.7%	637 ms
Inception_V4_quant	41 Mb	79.5%	93.9%	1250.8 ms



```
Input: The movie was boring and i did not really enoyed it
Output:
   Negative: 0.76577926
   Positive: 0.23422068
---------
```

Please enter your movie review.



Warsaw

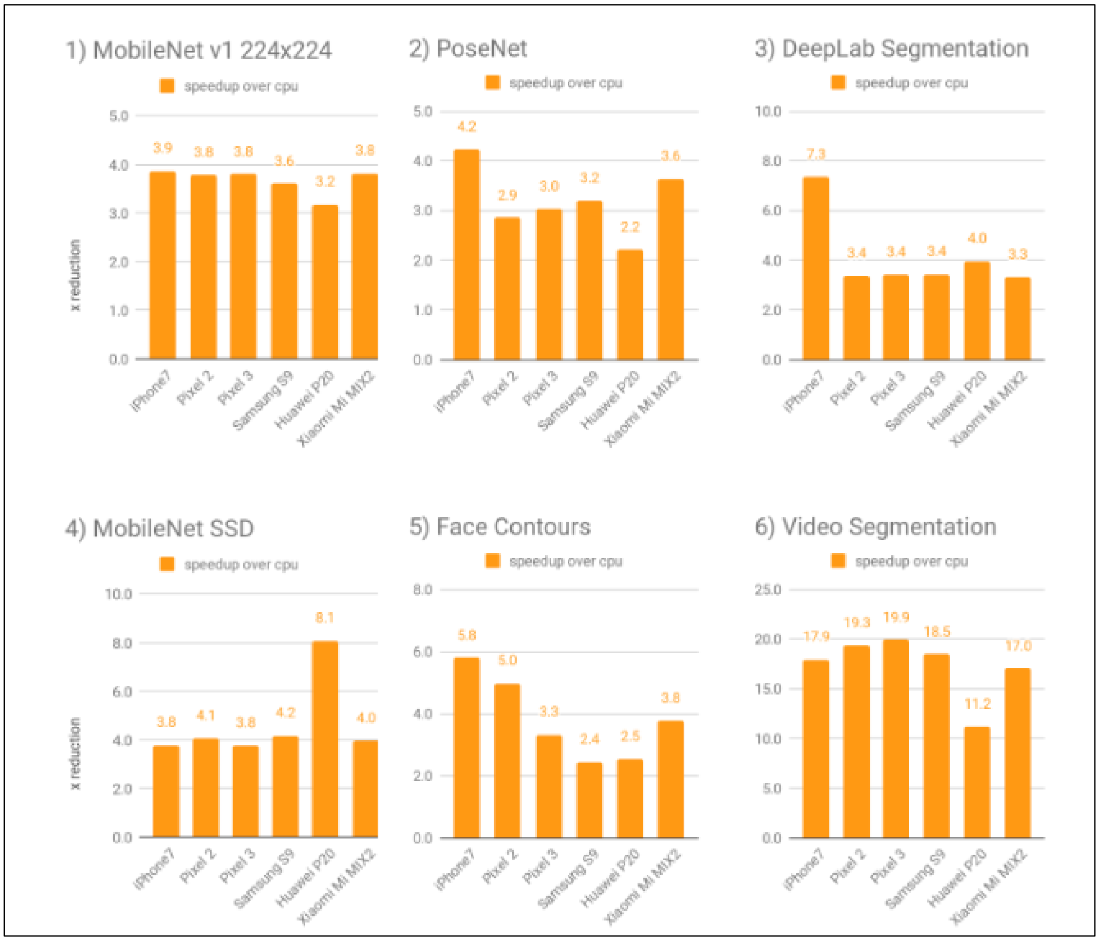
One of the most famous people born in Warsaw was Maria Skłodowska-Curie, who achieved international recognition for her research on radioactivity and was the first female recipient of the Nobel Prize. Famous musicians include Władysław Szpilman and Frédéric Chopin. Though Chopin was born in the village of Żelazowa Wola, about 60 km (37 mi) from Warsaw, he moved to the city with his family when he was seven months old. Casimir Pulaski, a Polish general and hero of the American Revolutionary War, was born here in 1745.

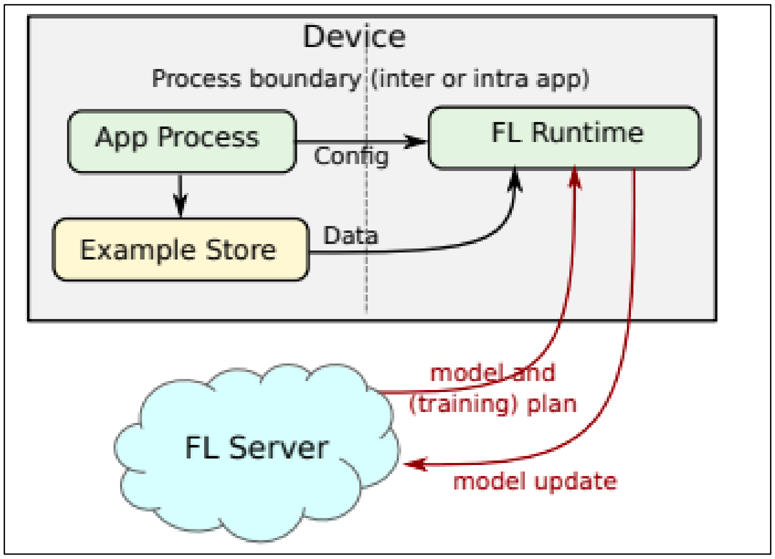
You might want to ask?

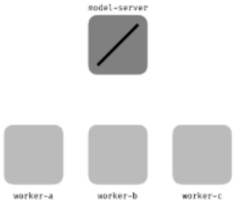
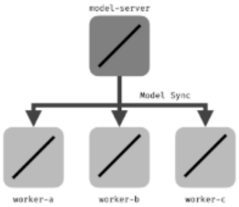
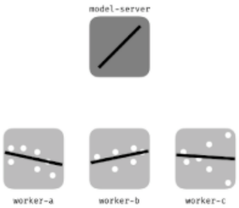
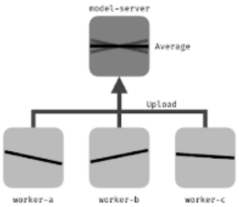
What was Maria Curie the first female recipient of?

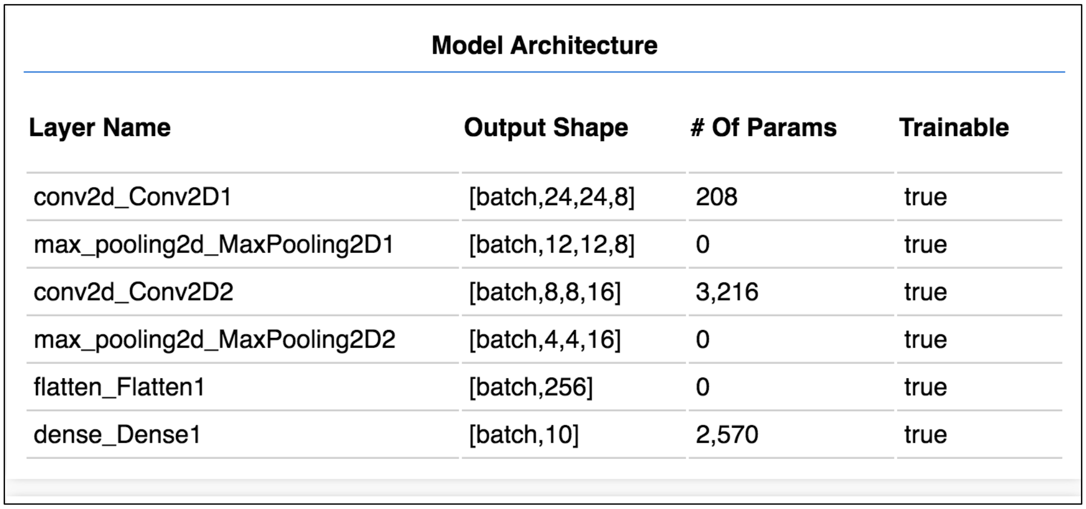
Text query

where marie curie was born?


1) MobileNet v1 224x224
speedup over cpu
x reduction
0.0
1.0
2.0
3.0
4.0
5.0
3.9
3.8
3.8
3.6
3.2
3.8
iPhone7
Pixel 2
Pixel 3
Samsung S9
Huawei P20
Xiaomi Mi MIX2
2) PoseNet
speedup over cpu
0.0
1.0
2.0
3.0
4.0
5.0
4.2
2.9
3.0
3.2
2.2
3.6
iPhone7
Pixel 2
Pixel 3
Samsung S9
Huawei P20
Xiaomi Mi MIX2
3) DeepLab Segmentation
speedup over cpu
0.0
2.0
4.0
6.0
8.0
10.0
7.3
3.4
3.4
3.4
4.0
3.3
iPhone7
Pixel 2
Pixel 3
Samsung S9
Huawei P20
Xiaomi Mi MIX2
4) MobileNet SSD
speedup over cpu
x reduction
0.0
2.0
4.0
6.0
8.0
10.0
3.8
4.1
3.8
4.2
8.1
4.0
iPhone7
Pixel 2
Pixel 3
Samsung S9
Huawei P20
Xiaomi Mi MIX2
5) Face Contours
speedup over cpu
0.0
2.0
4.0
6.0
8.0
5.8
5.0
3.3
2.4
2.5
3.8
iPhone7
Pixel 2
Pixel 3
Samsung S9
Huawei P20
Xiaomi Mi MIX2
6) Video Segmentation
speedup over cpu
0.0
5.0
10.0
15.0
20.0
25.0
17.9
19.3
19.9
18.5
11.2
17.0
iPhone7
Pixel 2
Pixel 3
Samsung S9
Huawei P20
Xiaomi Mi MIX2

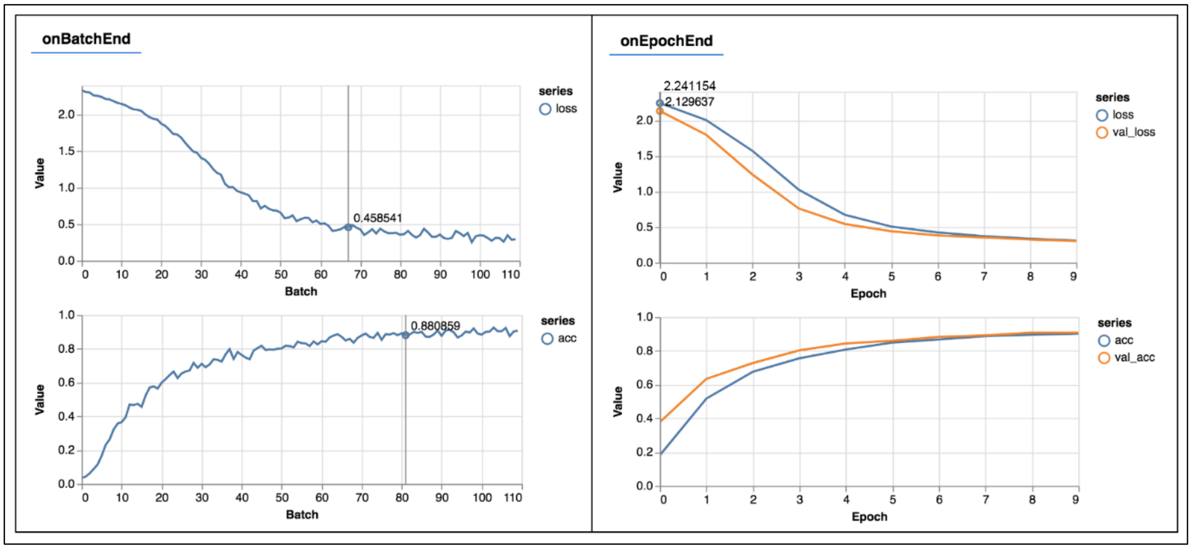


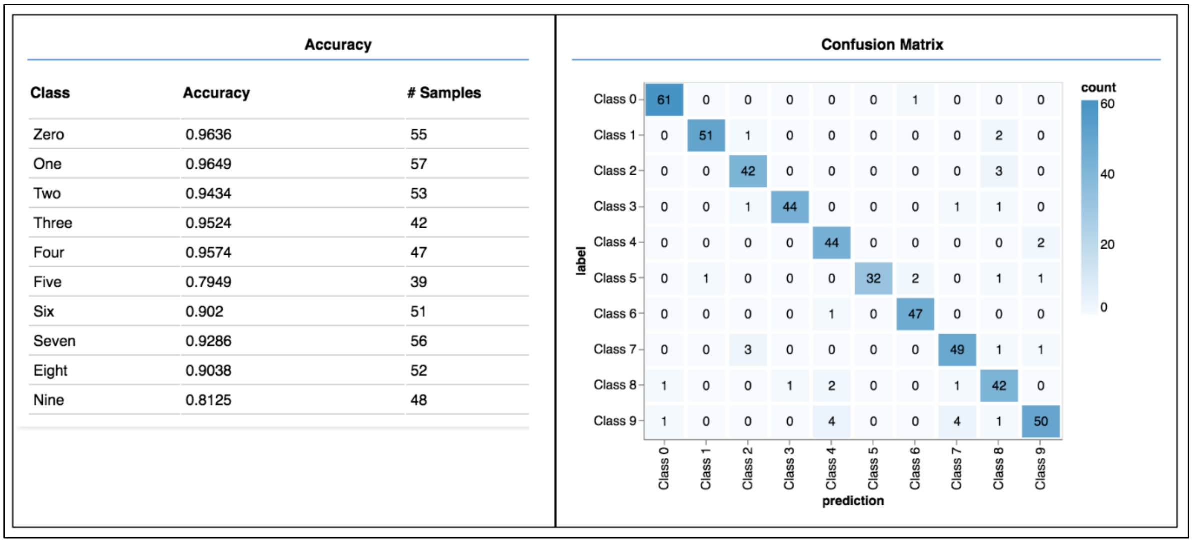
Step 1	Step 2	Step 3	Step 4
	 model-server Model Sync worker-a worker-b worker-c		 model-server Average Upload worker-a worker-b worker-c
Central server chooses a statistical model to be trained	Central server transmits the initial model to several nodes	Nodes train the model locally with their own data	Central server pools model results and generate one global mode without accessing any data



Model Architecture

Layer Name	Output Shape	# Of Params	Trainable
conv2d_Conv2D1	[batch,24,24,8]	208	true
max_pooling2d_MaxPooling2D1	[batch,12,12,8]	0	true
conv2d_Conv2D2	[batch,8,8,16]	3,216	true
max_pooling2d_MaxPooling2D2	[batch,4,4,16]	0	true
flatten_Flatten1	[batch,256]	0	true
dense_Dense1	[batch,10]	2,570	true





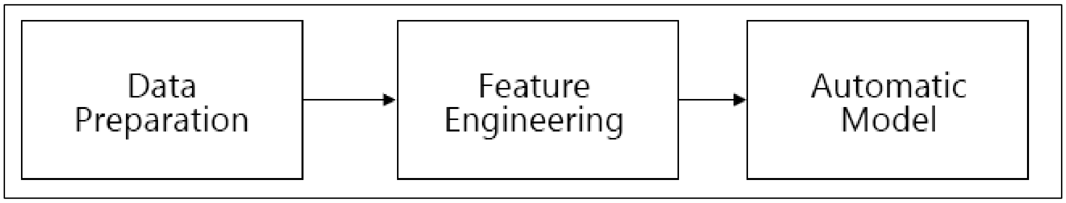
Accuracy

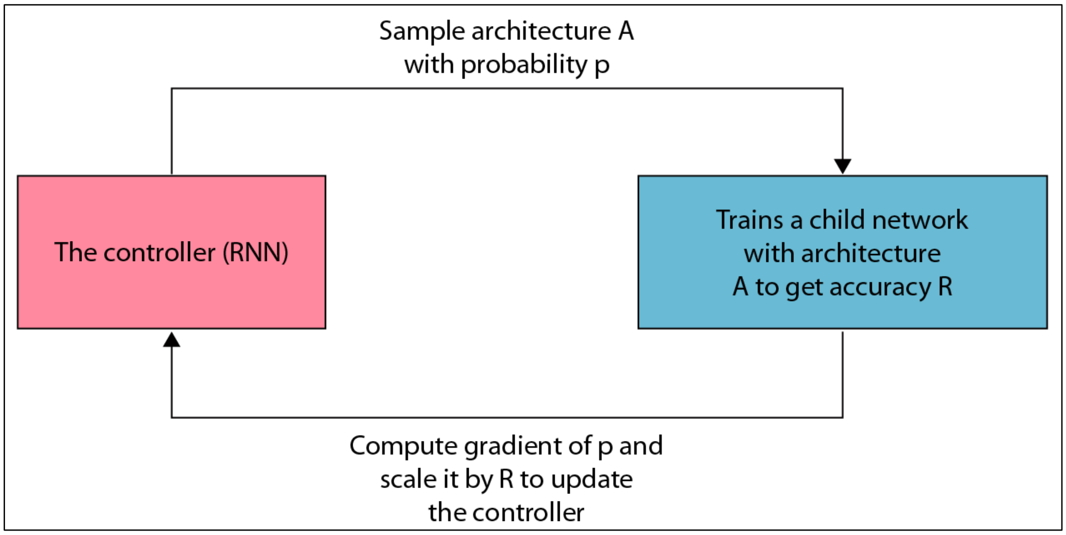
Class	Accuracy	# Samples
Zero	0.9636	55
One	0.9649	57
Two	0.9434	53
Three	0.9524	42
Four	0.9574	47
Five	0.7949	39
Six	0.902	51
Seven	0.9286	56
Eight	0.9038	52
Nine	0.8125	48

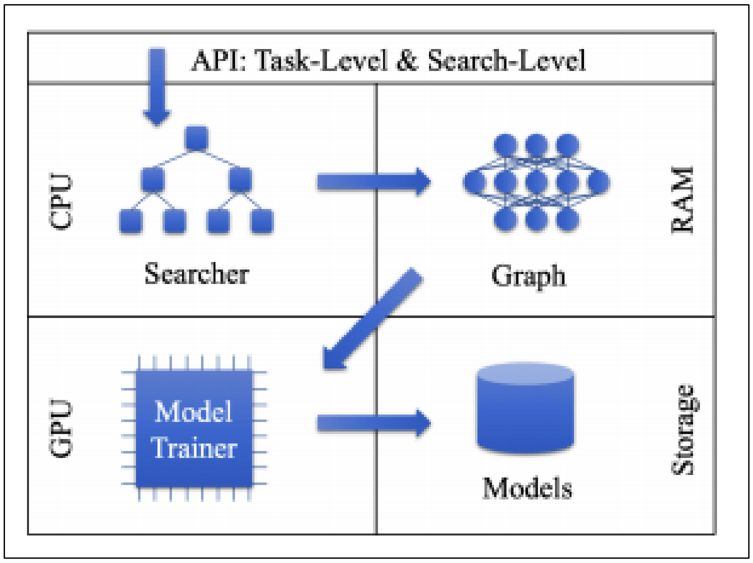
Confusion Matrix

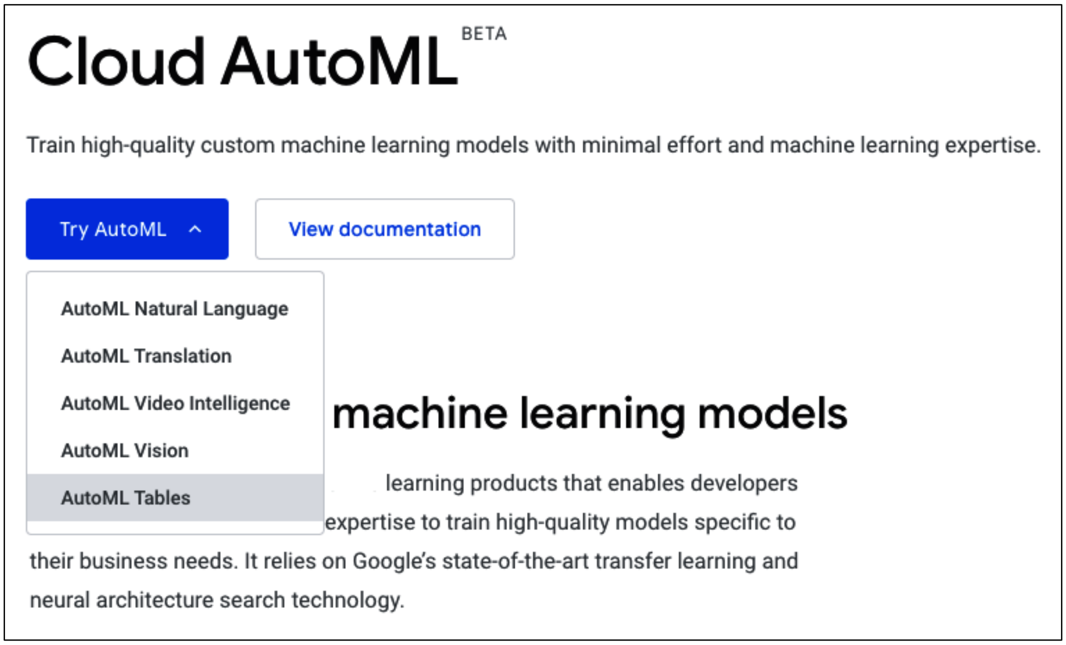
	Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9
Class 0	61	0	0	0	0	0	1	0	0	0
Class 1	0	51	1	0	0	0	0	0	2	0
Class 2	0	0	42	0	0	0	0	0	3	0
Class 3	0	0	1	44	0	0	0	1	1	0
Class 4	0	0	0	0	44	0	0	0	0	2
Class 5	0	1	0	0	0	32	2	0	1	1
Class 6	0	0	0	0	1	0	47	0	0	0
Class 7	0	0	3	0	0	0	0	49	1	1
Class 8	1	0	0	1	2	0	0	1	42	0
Class 9	1	0	0	0	4	0	0	4	1	50

Chapter 14: An introduction to AutoML




Sample architecture A with probability p
The controller (RNN)
Trains a child network with architecture A to get accuracy R
Compute gradient of p and scale it by R to update the controller



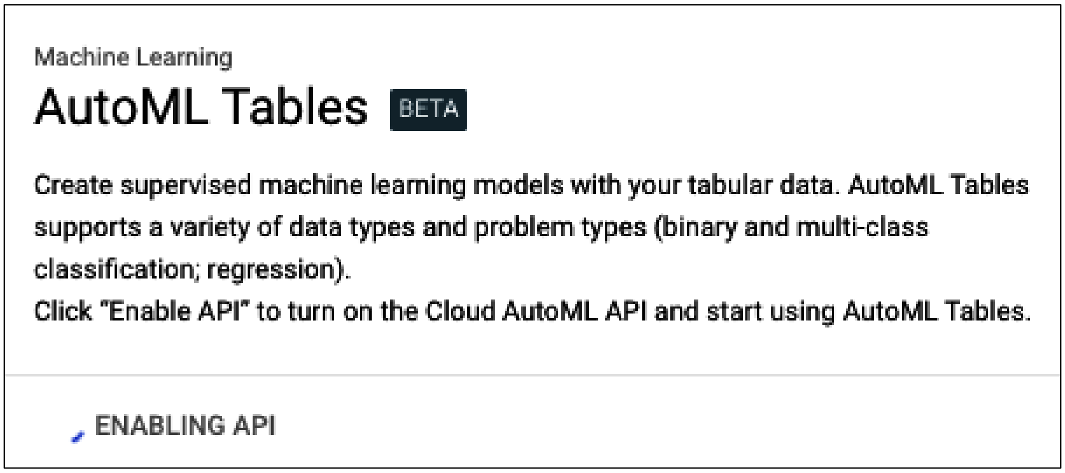


Cloud AutoML BETA

Train high-quality custom machine learning models with minimal effort and machine learning expertise.

machine learning models

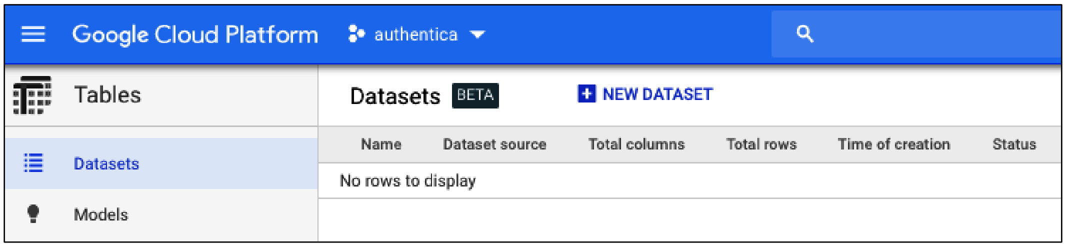
learning products that enables developers expertise to train high-quality models specific to their business needs. It relies on Google's state-of-the-art transfer learning and neural architecture search technology.



Machine Learning

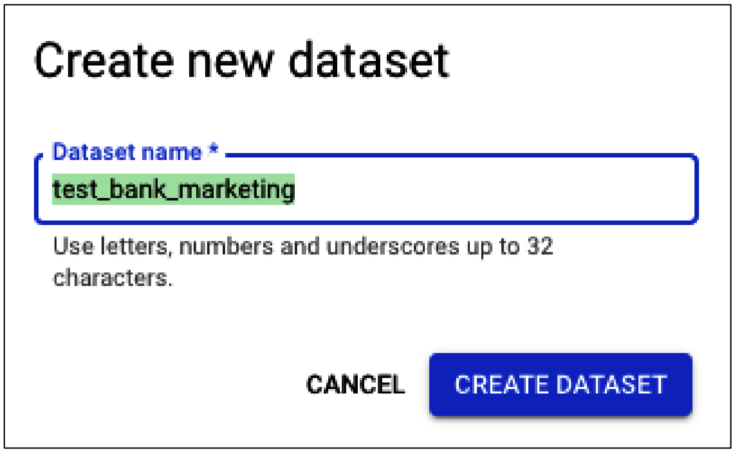
AutoML Tables BETA

Create supervised machine learning models with your tabular data. AutoML Tables supports a variety of data types and problem types (binary and multi-class classification; regression).
Click "Enable API" to turn on the Cloud AutoML API and start using AutoML Tables.



Datasets BETA

Name	Dataset source	Total columns	Total rows	Time of creation	Status
No rows to display					

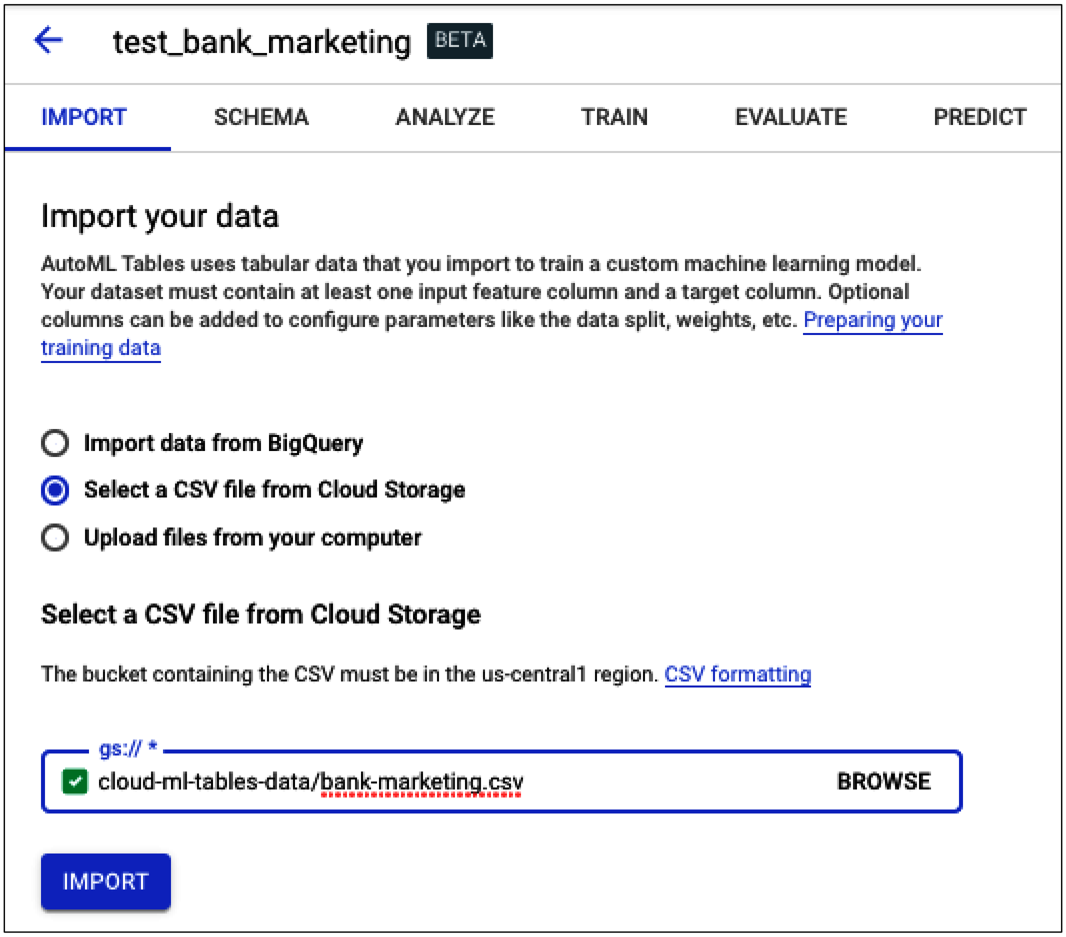


Create new dataset

Dataset name *

test_bank_marketing

Use letters, numbers and underscores up to 32 characters.



Import your data

AutoML Tables uses tabular data that you import to train a custom machine learning model. Your dataset must contain at least one input feature column and a target column. Optional columns can be added to configure parameters like the data split, weights, etc. Preparing your training data

○ Import data from BigQuery

● Select a CSV file from Cloud Storage

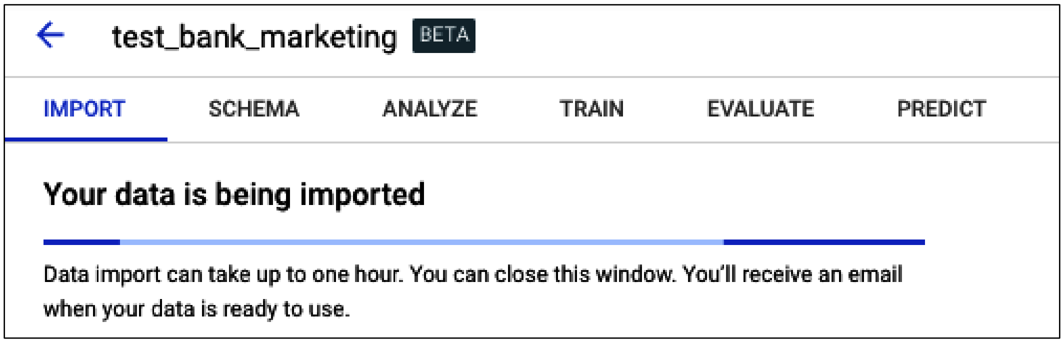
○ Upload files from your computer

Select a CSV file from Cloud Storage

The bucket containing the CSV must be in the us-central1 region. CSV formatting

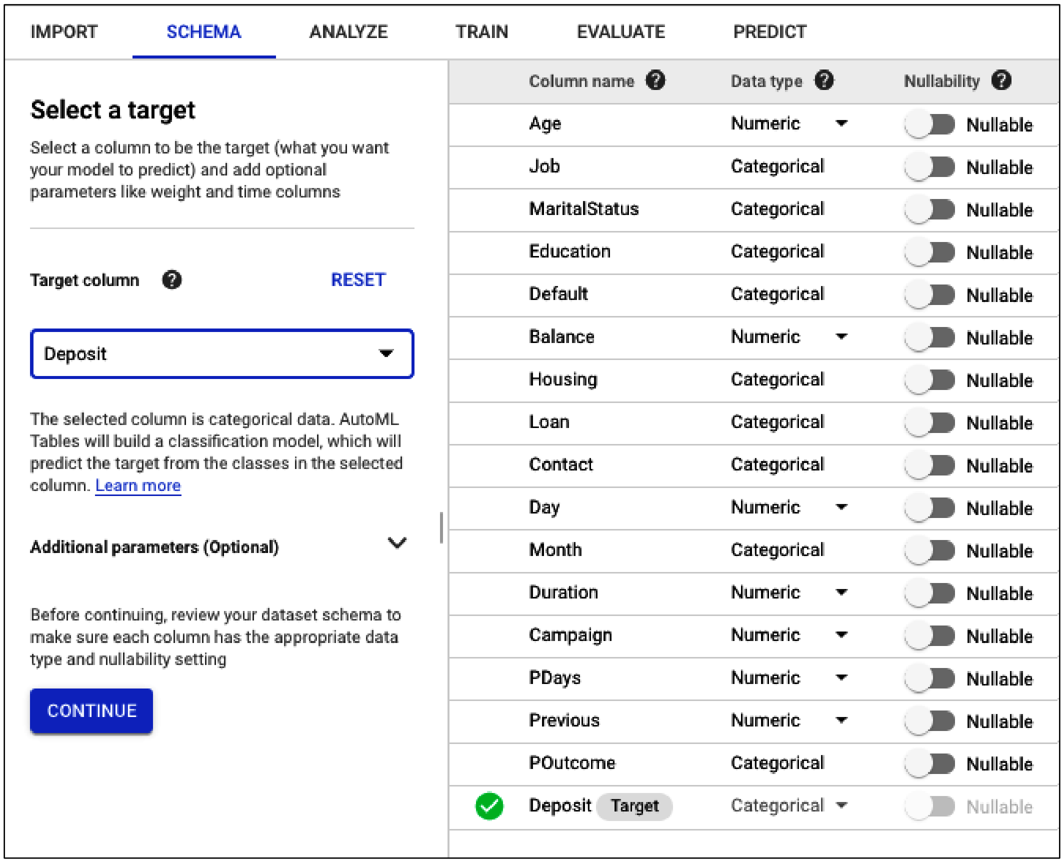
gs:// *

✓ cloud-ml-tables-data/bank-marketing.csv



Your data is being imported

Data import can take up to one hour. You can close this window. You'll receive an email when your data is ready to use.



Select a target

Select a column to be the target (what you want your model to predict) and add optional parameters like weight and time columns

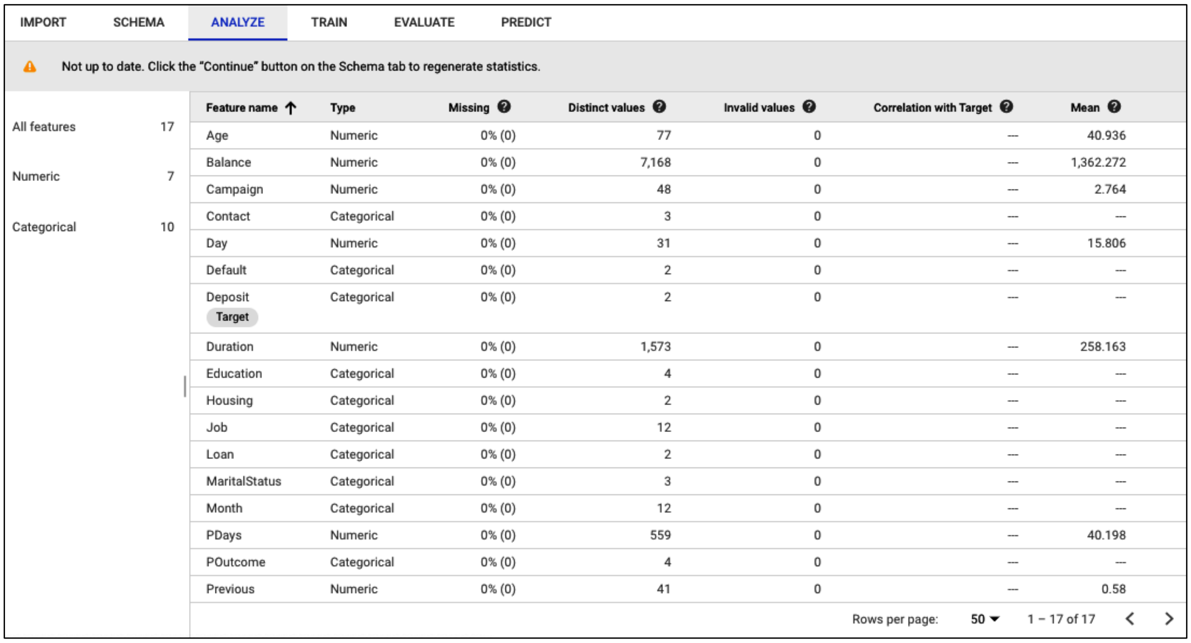
Target column: Deposit

The selected column is categorical data. AutoML Tables will build a classification model, which will predict the target from the classes in the selected column. Learn more

Additional parameters (Optional)

Before continuing, review your dataset schema to make sure each column has the appropriate data type and nullability setting

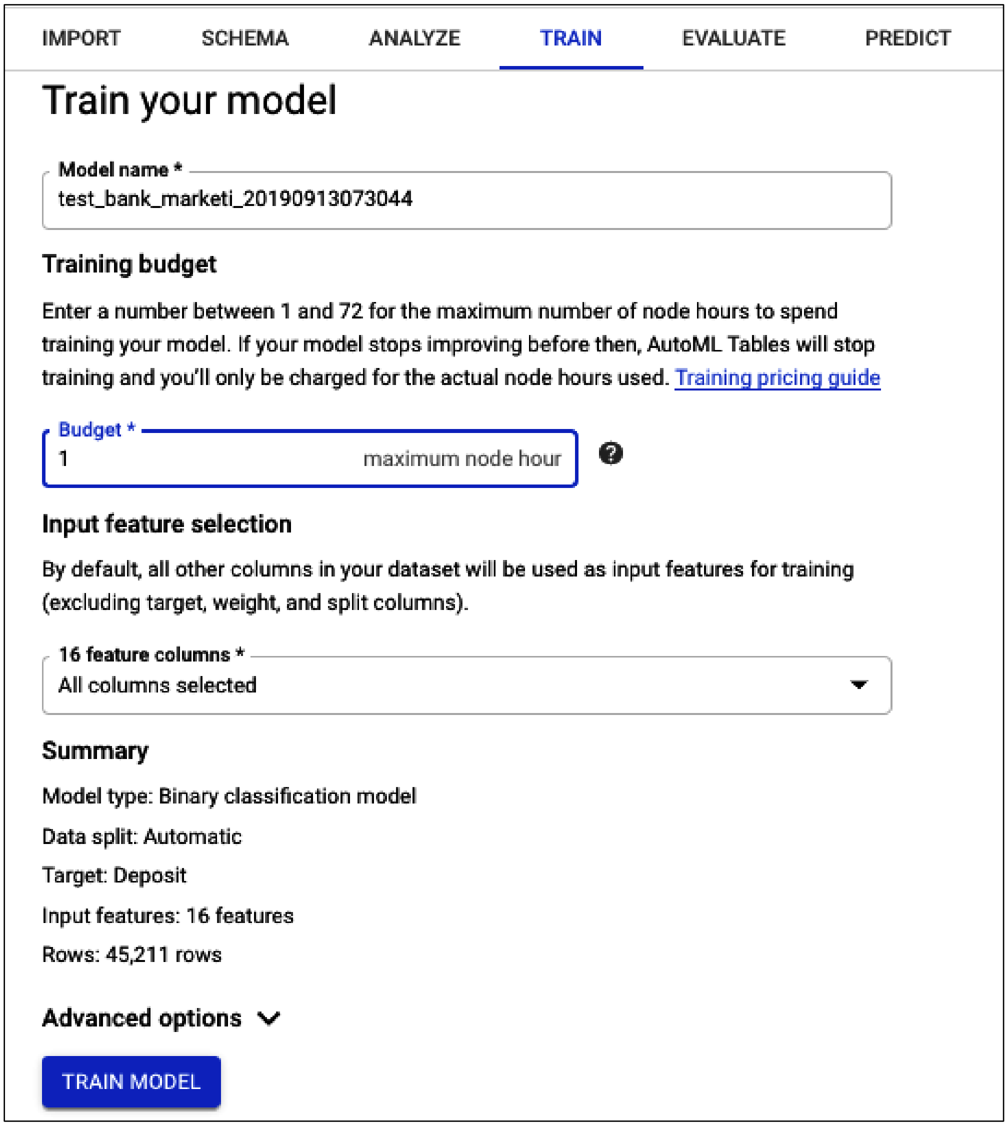
Column name	Data type	Nullability
Age	Numeric	Nullable
Job	Categorical	Nullable
MaritalStatus	Categorical	Nullable
Education	Categorical	Nullable
Default	Categorical	Nullable
Balance	Numeric	Nullable
Housing	Categorical	Nullable
Loan	Categorical	Nullable
Contact	Categorical	Nullable
Day	Numeric	Nullable
Month	Categorical	Nullable
Duration	Numeric	Nullable
Campaign	Numeric	Nullable
PDays	Numeric	Nullable
Previous	Numeric	Nullable
POutcome	Categorical	Nullable
✓ Deposit Target	Categorical	Nullable



Not up to date. Click the "Continue" button on the Schema tab to regenerate statistics.

All features	17
Numeric	7
Categorical	10

Feature name ↑	Type	Missing	Distinct values	Invalid values	Correlation with Target	Mean
Age	Numeric	0% (0)	77	0	—	40.936
Balance	Numeric	0% (0)	7,168	0	—	1,362.272
Campaign	Numeric	0% (0)	48	0	—	2.764
Contact	Categorical	0% (0)	3	0	—	—
Day	Numeric	0% (0)	31	0	—	15.806
Default	Categorical	0% (0)	2	0	—	—
Deposit Target	Categorical	0% (0)	2	0	—	—
Duration	Numeric	0% (0)	1,573	0	—	258.163
Education	Categorical	0% (0)	4	0	—	—
Housing	Categorical	0% (0)	2	0	—	—
Job	Categorical	0% (0)	12	0	—	—
Loan	Categorical	0% (0)	2	0	—	—
MaritalStatus	Categorical	0% (0)	3	0	—	—
Month	Categorical	0% (0)	12	0	—	—
PDays	Numeric	0% (0)	559	0	—	40.198
POutcome	Categorical	0% (0)	4	0	—	—
Previous	Numeric	0% (0)	41	0	—	0.58



Train your model

Model name *
test_bank_marketi_20190913073044

Training budget

Enter a number between 1 and 72 for the maximum number of node hours to spend training your model. If your model stops improving before then, AutoML Tables will stop training and you'll only be charged for the actual node hours used. Training pricing guide

Budget *
1 maximum node hour

Input feature selection

By default, all other columns in your dataset will be used as input features for training (excluding target, weight, and split columns).

16 feature columns *
All columns selected

Summary

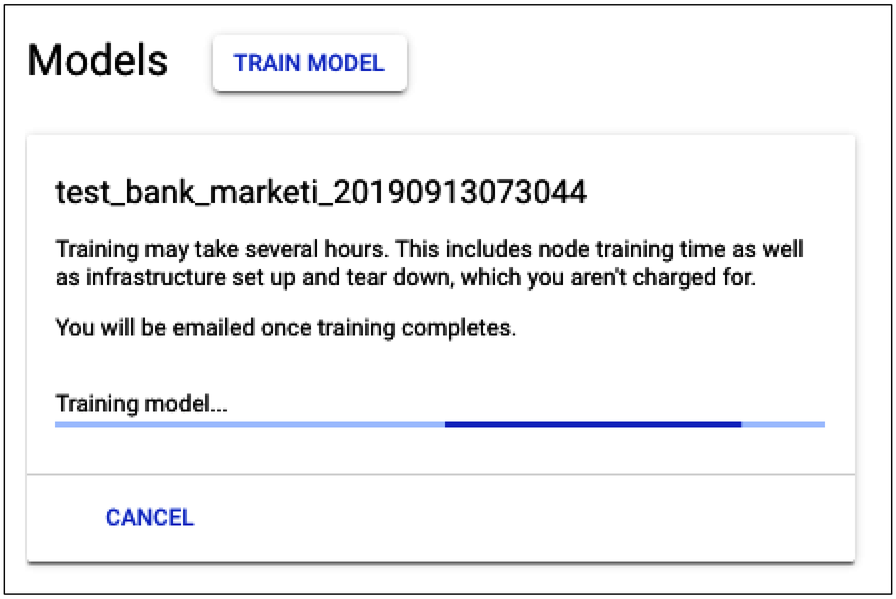
Model type: Binary classification model

Data split: Automatic

Target: Deposit

Input features: 16 features

Rows: 45,211 rows

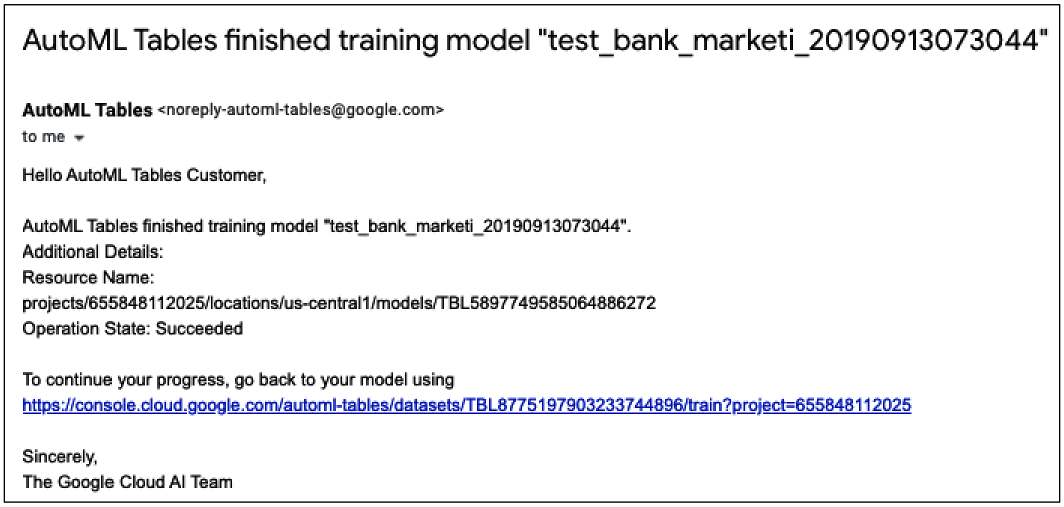


test_bank_marketi_20190913073044

Training may take several hours. This includes node training time as well as infrastructure set up and tear down, which you aren't charged for.

You will be emailed once training completes.

Training model...



AutoML Tables finished training model "test_bank_marketi_20190913073044"

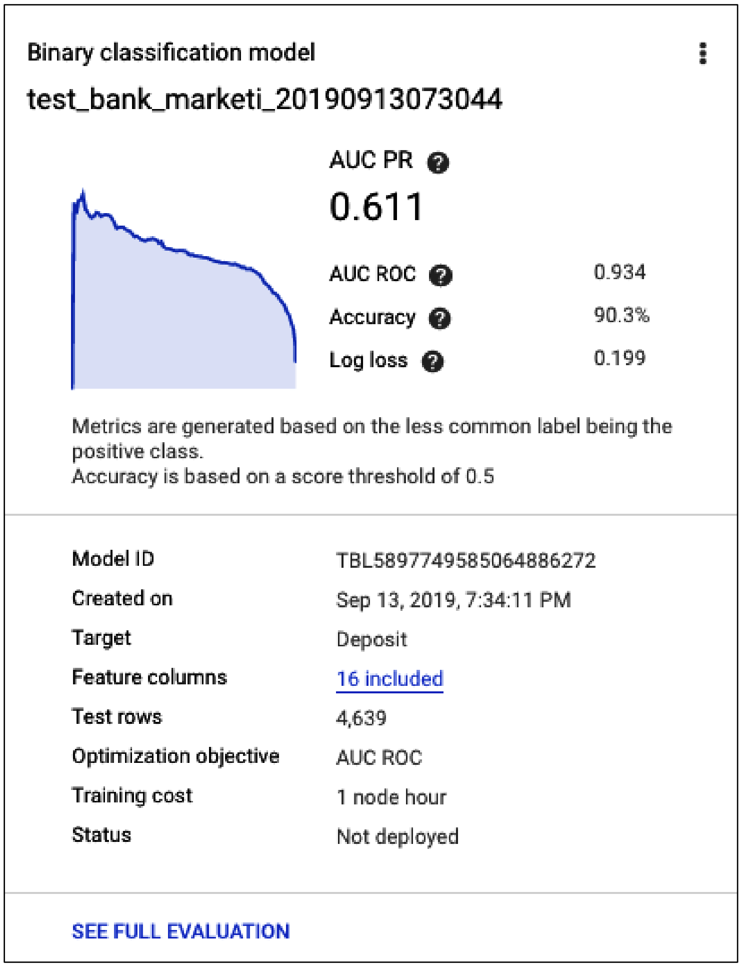
AutoML Tables <noreply-automl-tables@google.com>
to me

Hello AutoML Tables Customer,

AutoML Tables finished training model "test_bank_marketi_20190913073044".
Additional Details:
Resource Name:
projects/655848112025/locations/us-central1/models/TBL5897749585064886272
Operation State: Succeeded

To continue your progress, go back to your model using
https://console.cloud.google.com/automl-tables/datasets/TBL8775197903233744896/train?project=655848112025

Sincerely,
The Google Cloud AI Team



Binary classification model

test_bank_marketi_20190913073044

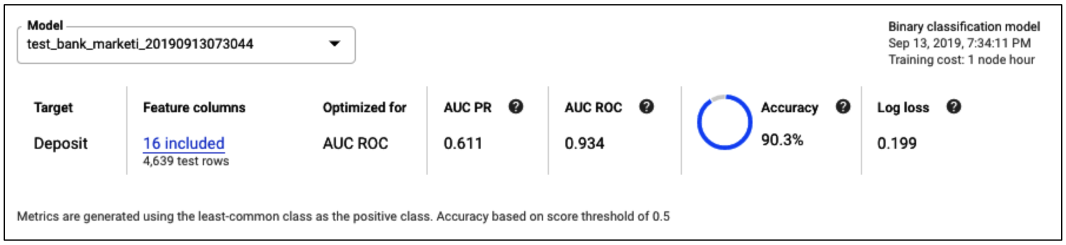
AUC PR ?

0.611

AUC ROC ?	0.934
Accuracy ?	90.3%
Log loss ?	0.199

Metrics are generated based on the less common label being the positive class.
Accuracy is based on a score threshold of 0.5

Model ID	TBL5897749585064886272
Created on	Sep 13, 2019, 7:34:11 PM
Target	Deposit
Feature columns	16 included
Test rows	4,639
Optimization objective	AUC ROC
Training cost	1 node hour
Status	Not deployed

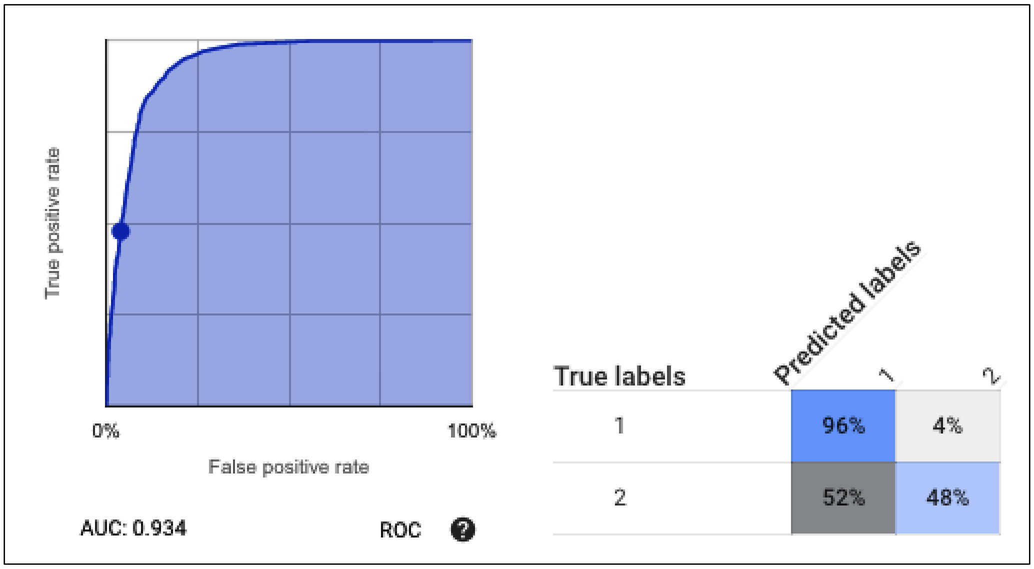


Model
test_bank_marketi_20190913073044

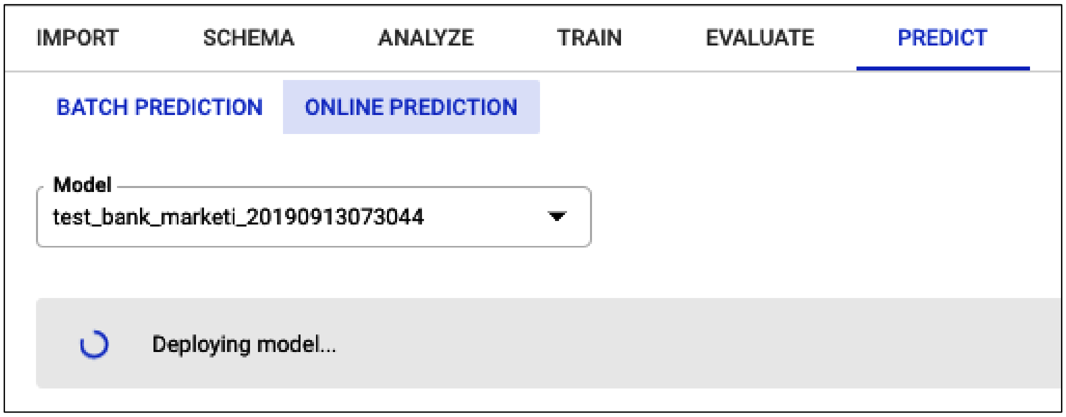
Binary classification model
Sep 13, 2019, 7:34:11 PM
Training cost: 1 node hour

Target	Feature columns	Optimized for	AUC PR ?	AUC ROC ?	Accuracy ?	Log loss ?
Deposit	16 included 4,639 test rows	AUC ROC	0.611	0.934	90.3%	0.199

Metrics are generated using the least-common class as the positive class. Accuracy based on score threshold of 0.5

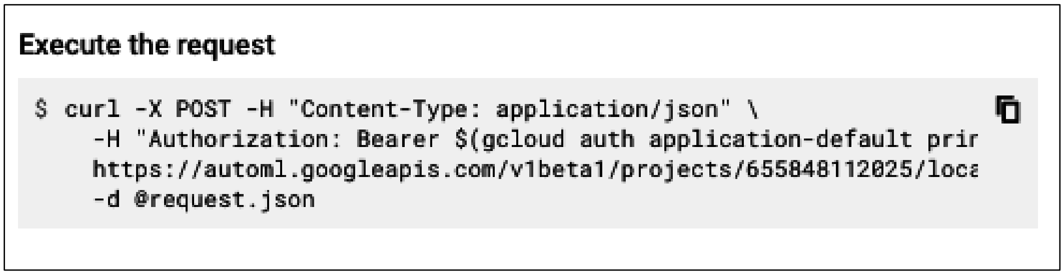


True labels	1	2
1	96%	4%
2	52%	48%



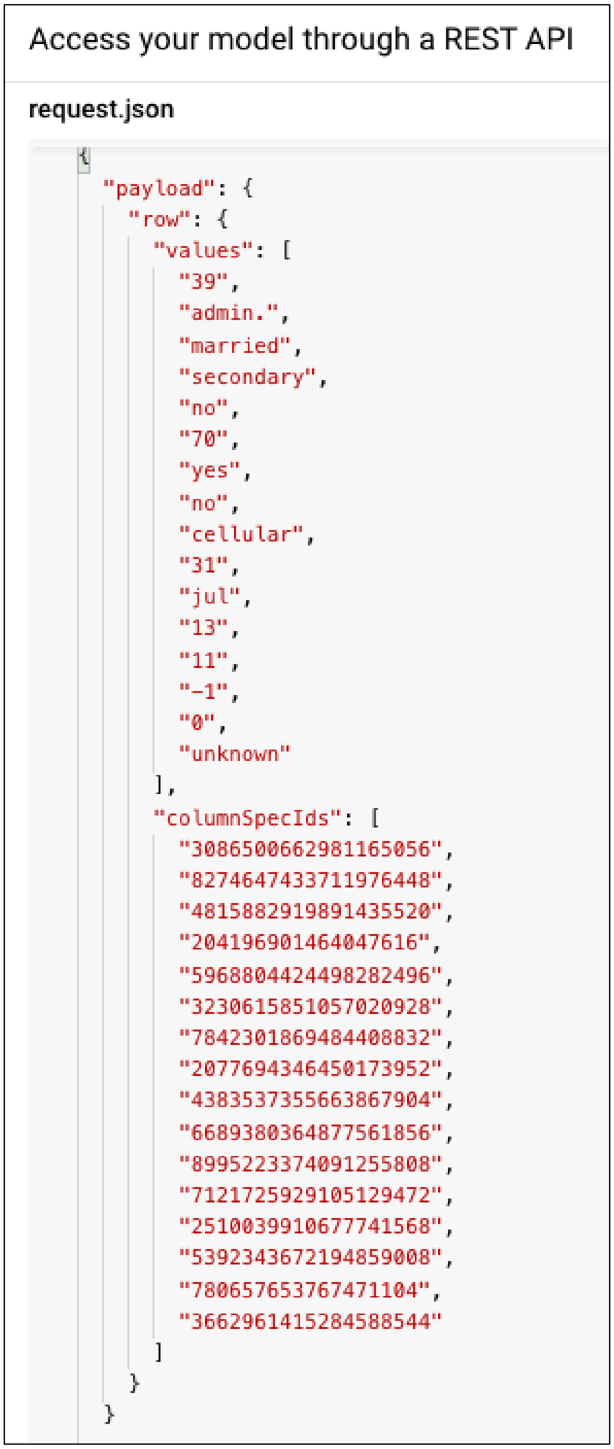
Model: test_bank_marketi_20190913073044

Deploying model...



Execute the request

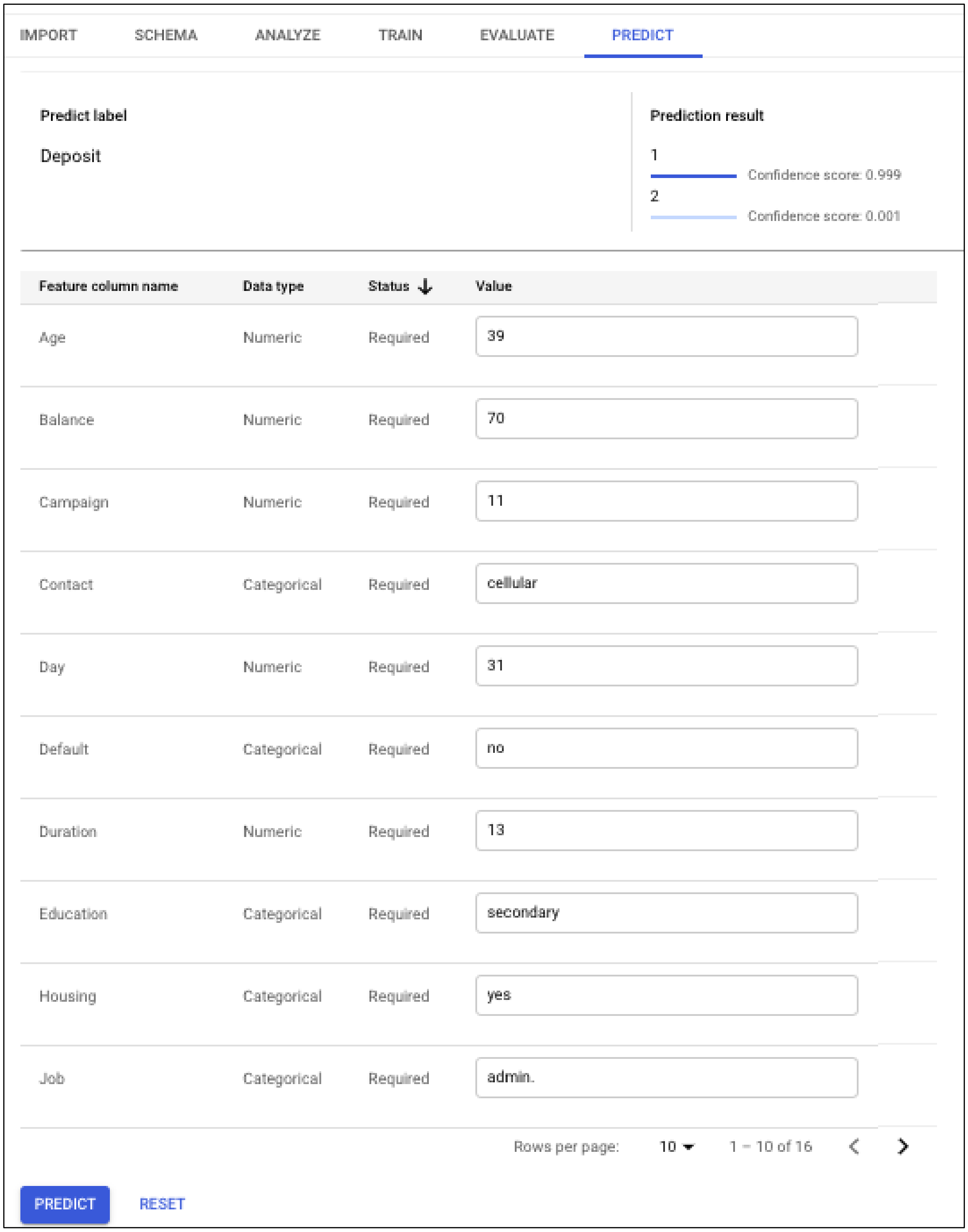
```
$ curl -X POST -H "Content-Type: application/json" \
    -H "Authorization: Bearer $(gcloud auth application-default prir
    https://automl.googleapis.com/v1beta1/projects/655848112025/loca
    -d @request.json
```



Access your model through a REST API

request.json

```
{
  "payload": {
    "row": {
      "values": [
        "39",
        "admin.",
        "married",
        "secondary",
        "no",
        "70",
        "yes",
        "no",
        "cellular",
        "31",
        "jul",
        "13",
        "11",
        "-1",
        "0",
        "unknown"
      ],
      "columnSpecIds": [
        "3086500662981165056",
        "8274647433711976448",
        "4815882919891435520",
        "204196901464047616",
        "5968804424498282496",
        "3230615851057020928",
        "7842301869484408832",
        "2077694346450173952",
        "4383537355663867904",
        "6689380364877561856",
        "8995223374091255808",
        "7121725929105129472",
        "2510039910677741568",
        "5392343672194859008",
        "780657653767471104",
        "3662961415284588544"
      ]
    }
  }
```



Predict label

Deposit

Prediction result

1 Confidence score: 0.999

2 Confidence score: 0.001

Feature column name	Data type	Status ↓	Value
Age	Numeric	Required	39
Balance	Numeric	Required	70
Campaign	Numeric	Required	11
Contact	Categorical	Required	cellular
Day	Numeric	Required	31
Default	Categorical	Required	no
Duration	Numeric	Required	13
Education	Categorical	Required	secondary
Housing	Categorical	Required	yes
Job	Categorical	Required	admin.

Rows per page: 10 1 – 10 of 16

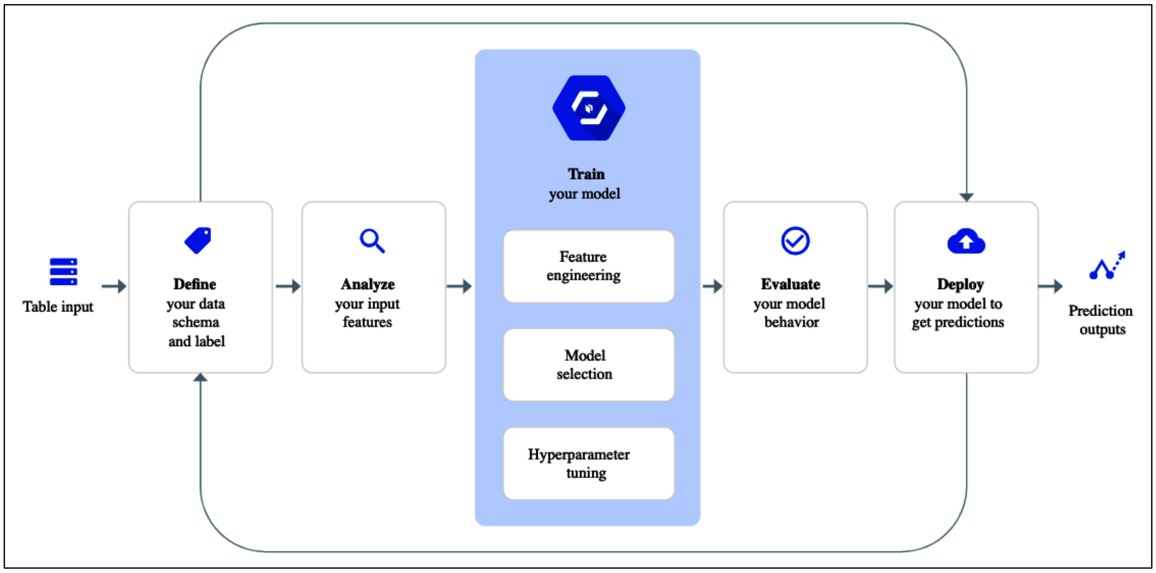
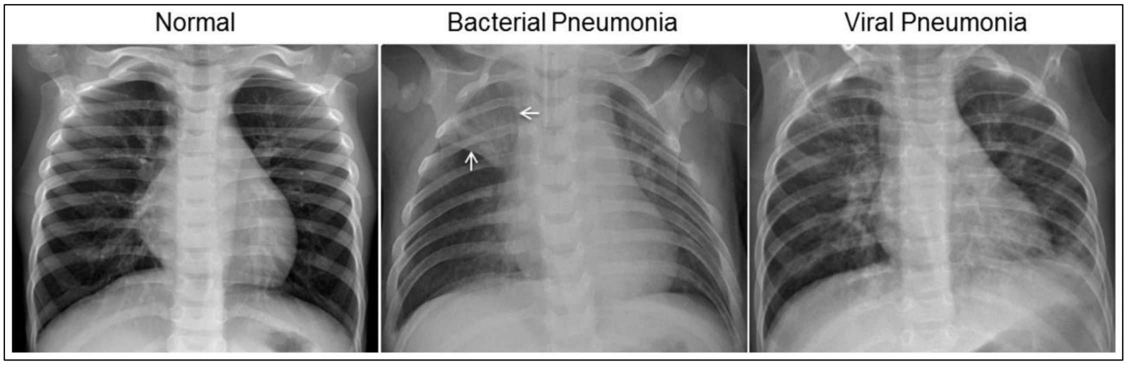
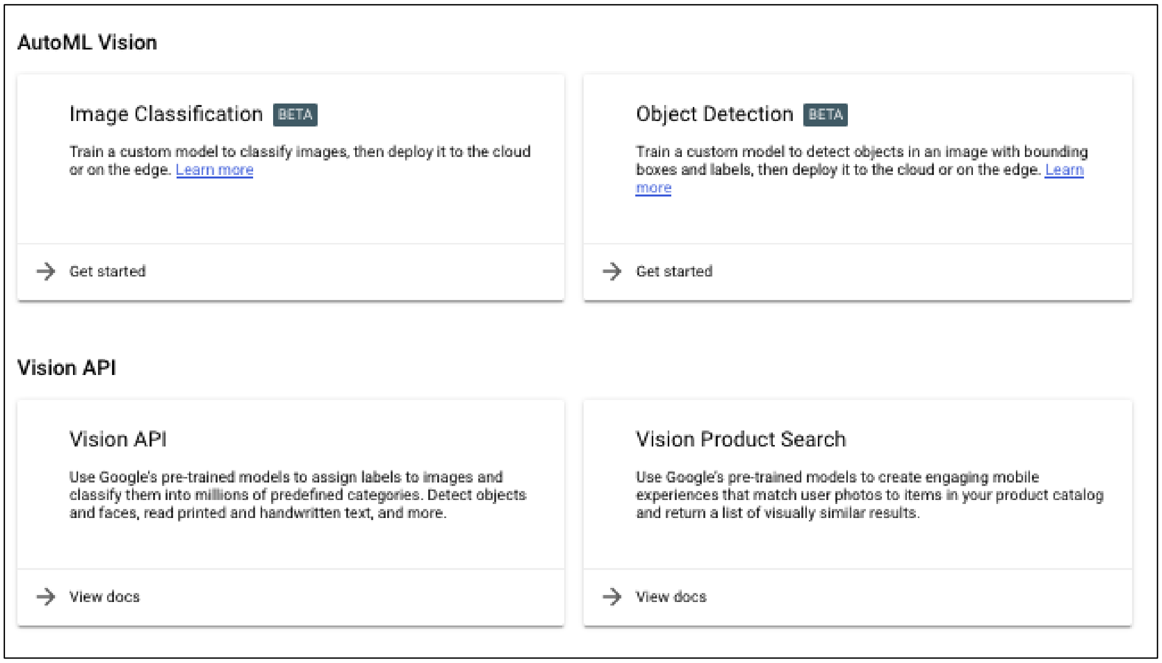


Table input
Define
your data
schema
and label
Analyze
your input
features
Train
your model
Feature
engineering
Model
selection
Hyperparameter
tuning
Evaluate
your model
behavior
Deploy
your model to
get predictions
Prediction
outputs


Normal
Bacterial Pneumonia
Viral Pneumonia



AutoML Vision

Image Classification BETA

Train a custom model to classify images, then deploy it to the cloud or on the edge. Learn more

→ Get started

Object Detection BETA

Train a custom model to detect objects in an image with bounding boxes and labels, then deploy it to the cloud or on the edge. Learn more

→ Get started

Vision API

Vision API

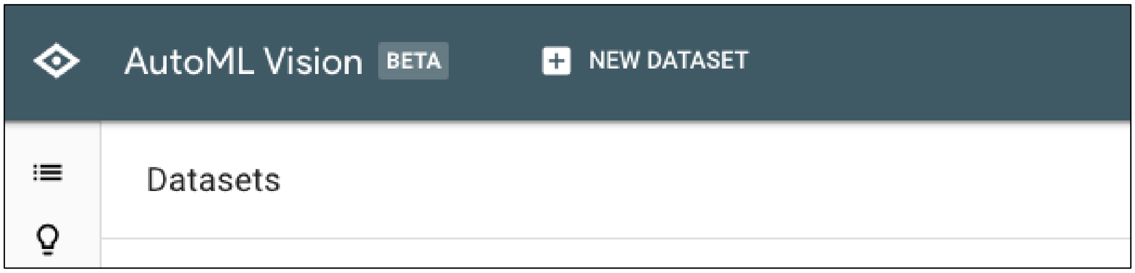
Use Google's pre-trained models to assign labels to images and classify them into millions of predefined categories. Detect objects and faces, read printed and handwritten text, and more.

→ View docs

Vision Product Search

Use Google's pre-trained models to create engaging mobile experiences that match user photos to items in your product catalog and return a list of visually similar results.

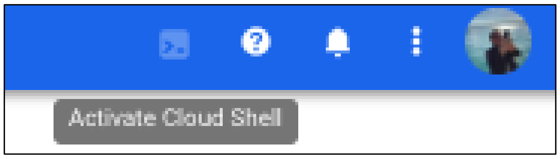
→ View docs

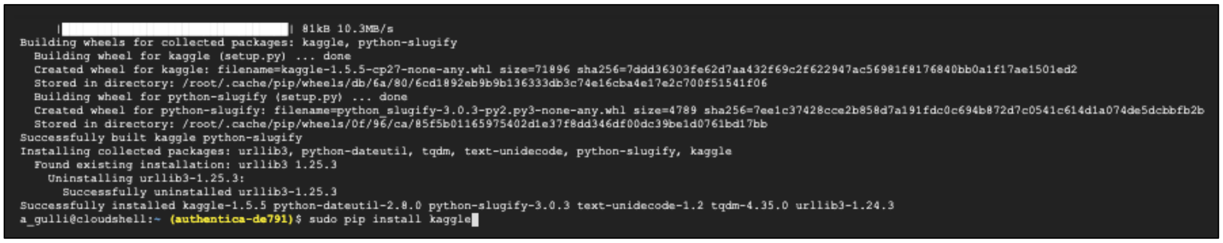


AutoML Vision BETA

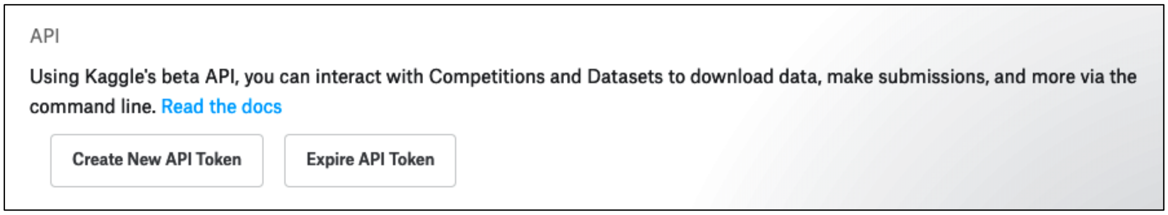
NEW DATASET

Datasets



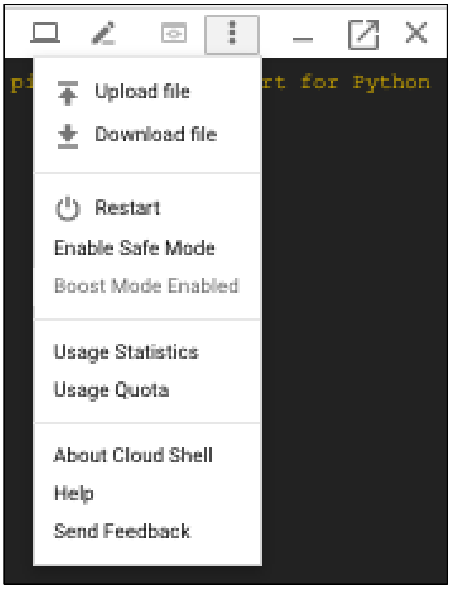


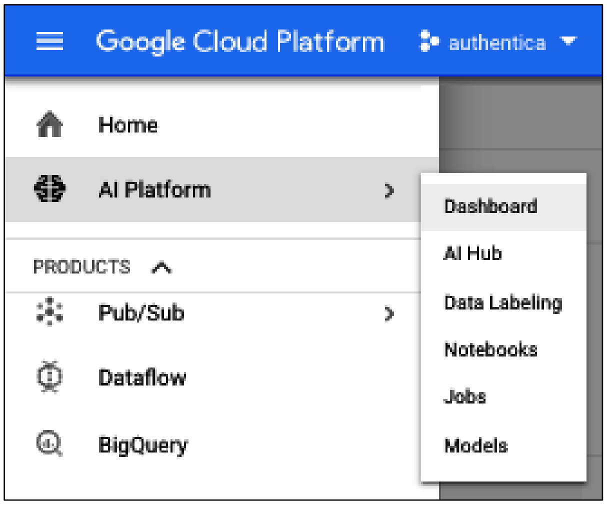
```
     |████████████████████████████████| 81kB 10.3MB/s
Building wheels for collected packages: kaggle, python-slugify
  Building wheel for kaggle (setup.py) ... done
  Created wheel for kaggle: filename=kaggle-1.5.5-cp27-none-any.whl size=71896 sha256=7ddd36303fe62d7aa432f69c2f622947ac56981f8176840bb0a1f17ae1501ed2
  Stored in directory: /root/.cache/pip/wheels/db/6a/80/6cd1892eb9b9b136333db3c74e16cba4e17e2c700f51541f06
  Building wheel for python-slugify (setup.py) ... done
  Created wheel for python-slugify: filename=python_slugify-3.0.3-py2.py3-none-any.whl size=4789 sha256=7ee1c37428cce2b858d7a191fdc0c694b872d7c0541c614d1a074de5dcbbfb2b
  Stored in directory: /root/.cache/pip/wheels/0f/96/ca/85f5b01165975402d1e37f8dd346df00dc39be1d0761bd17bb
Successfully built kaggle python-slugify
Installing collected packages: urllib3, python-dateutil, tqdm, text-unidecode, python-slugify, kaggle
  Found existing installation: urllib3 1.25.3
    Uninstalling urllib3-1.25.3:
      Successfully uninstalled urllib3-1.25.3
Successfully installed kaggle-1.5.5 python-dateutil-2.8.0 python-slugify-3.0.3 text-unidecode-1.2 tqdm-4.35.0 urllib3-1.24.3
a_gulli@cloudshell:~ (authentica-de791)$ sudo pip install kaggle
```

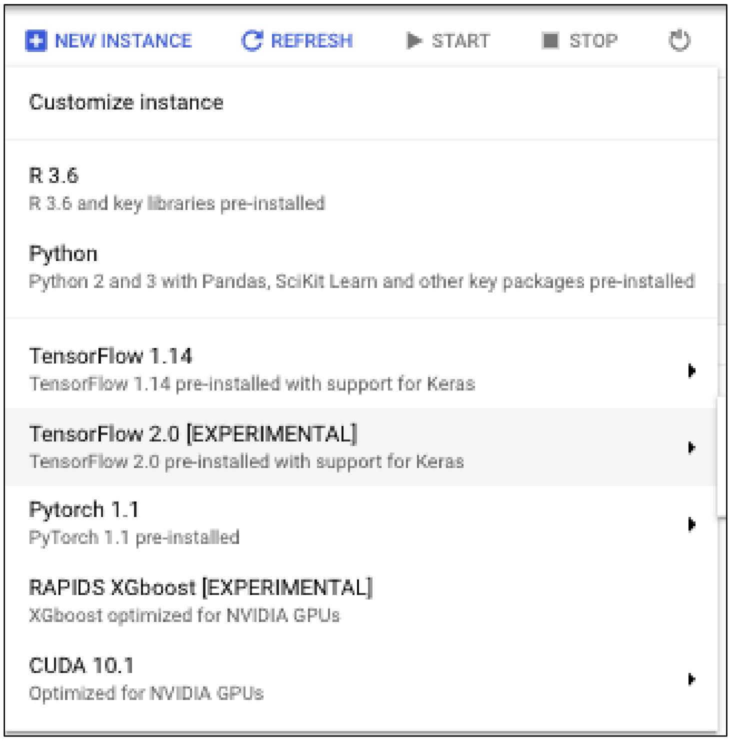


API

Using Kaggle's beta API, you can interact with Competitions and Datasets to download data, make submissions, and more via the command line. Read the docs







Customize instance

R 3.6
R 3.6 and key libraries pre-installed

Python
Python 2 and 3 with Pandas, SciKit Learn and other key packages pre-installed

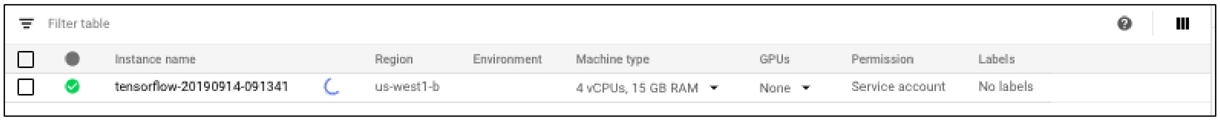
TensorFlow 1.14
TensorFlow 1.14 pre-installed with support for Keras

TensorFlow 2.0 [EXPERIMENTAL]
TensorFlow 2.0 pre-installed with support for Keras

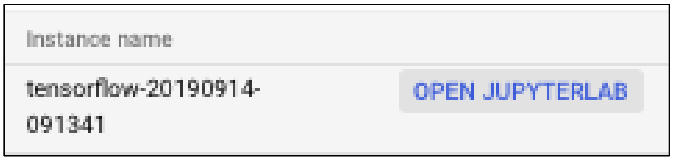
Pytorch 1.1
PyTorch 1.1 pre-installed

RAPIDS XGboost [EXPERIMENTAL]
XGboost optimized for NVIDIA GPUs

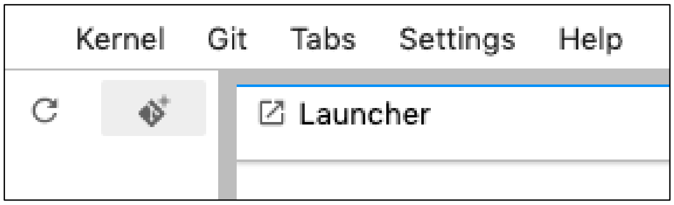
CUDA 10.1
Optimized for NVIDIA GPUs

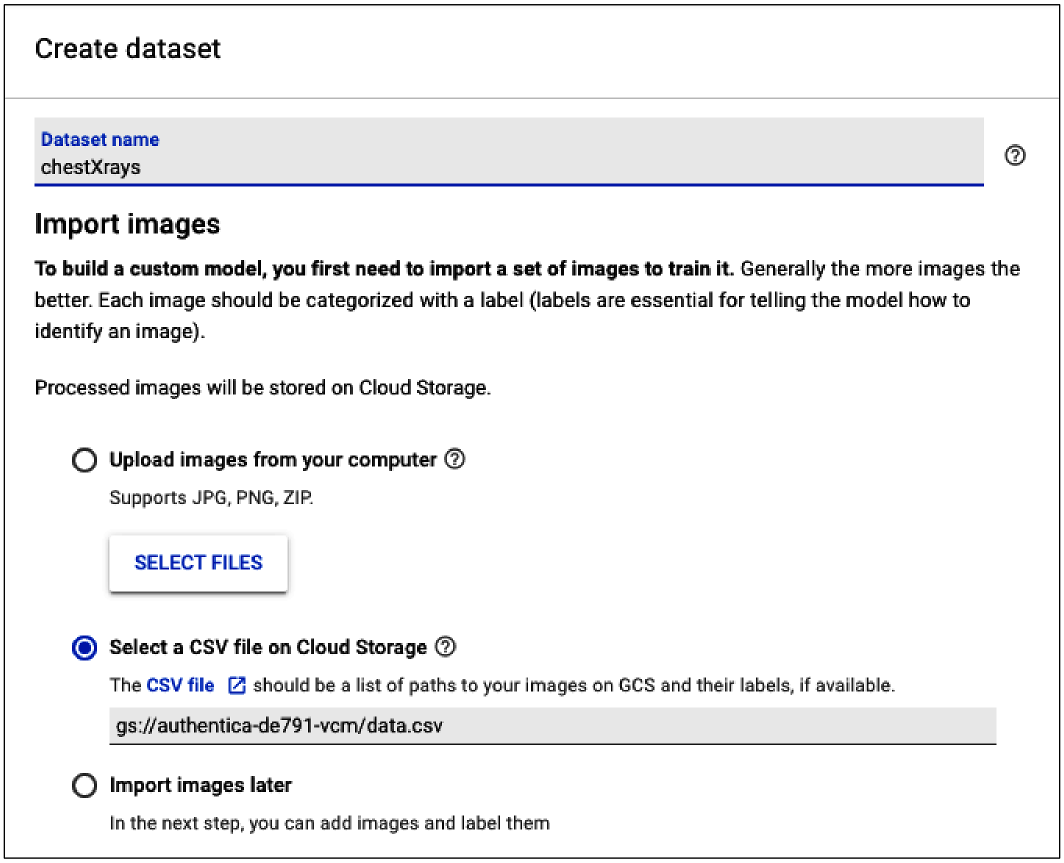


□	●	Instance name		Region	Environment	Machine type	GPUs	Permission	Labels
□	✓	tensorflow-20190914-091341		us-west1-b		4 vCPUs, 15 GB RAM ▾	None ▾	Service account	No labels


Instance name

tensorflow-20190914-091341





Create dataset

Dataset name
chestXrays

Import images

To build a custom model, you first need to import a set of images to train it. Generally the more images the better. Each image should be categorized with a label (labels are essential for telling the model how to identify an image).

Processed images will be stored on Cloud Storage.

○ **Upload images from your computer**

Supports JPG, PNG, ZIP.

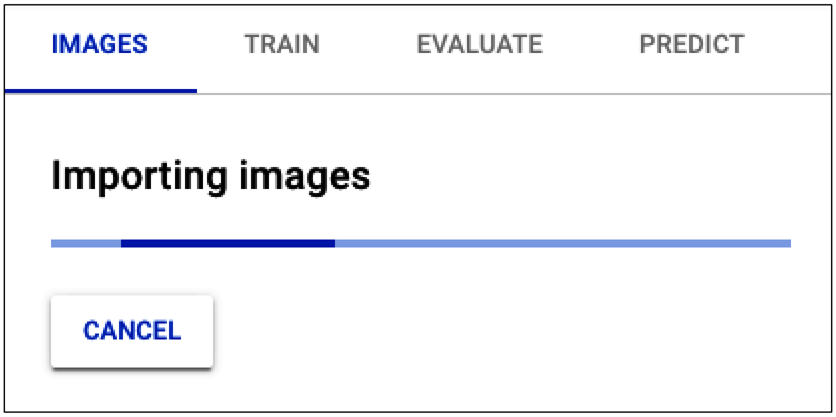
● **Select a CSV file on Cloud Storage**

The CSV file should be a list of paths to your images on GCS and their labels, if available.

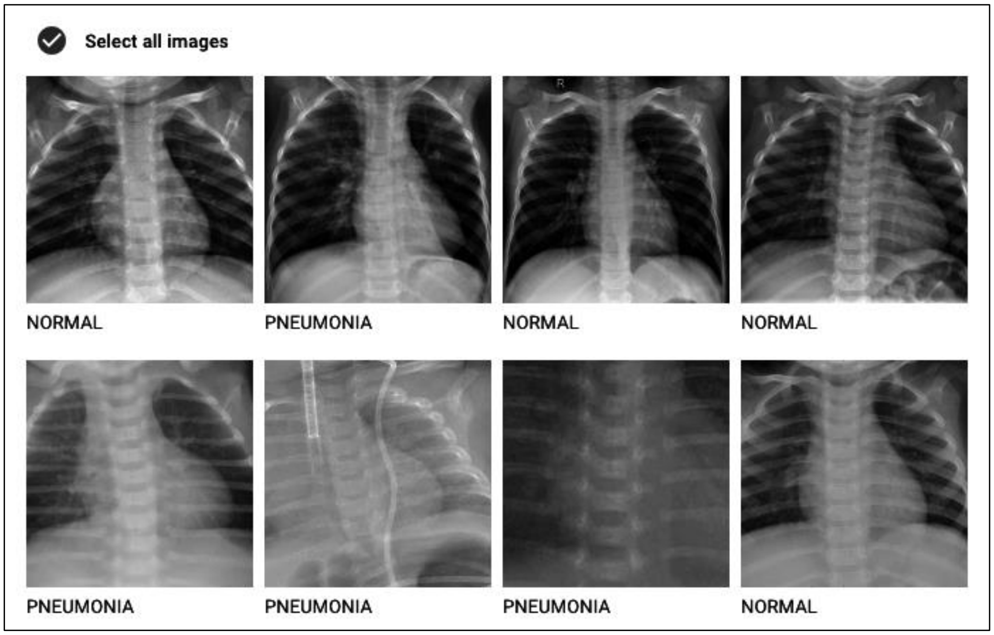
gs://authentica-de791-vcm/data.csv

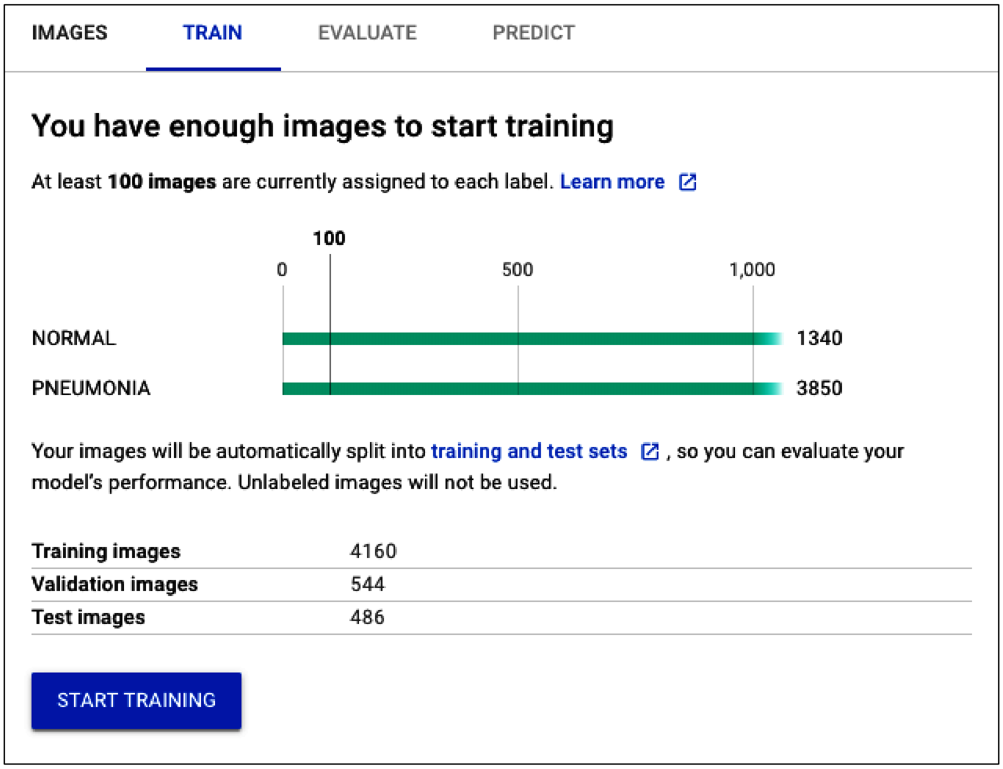
○ **Import images later**

In the next step, you can add images and label them



Importing images



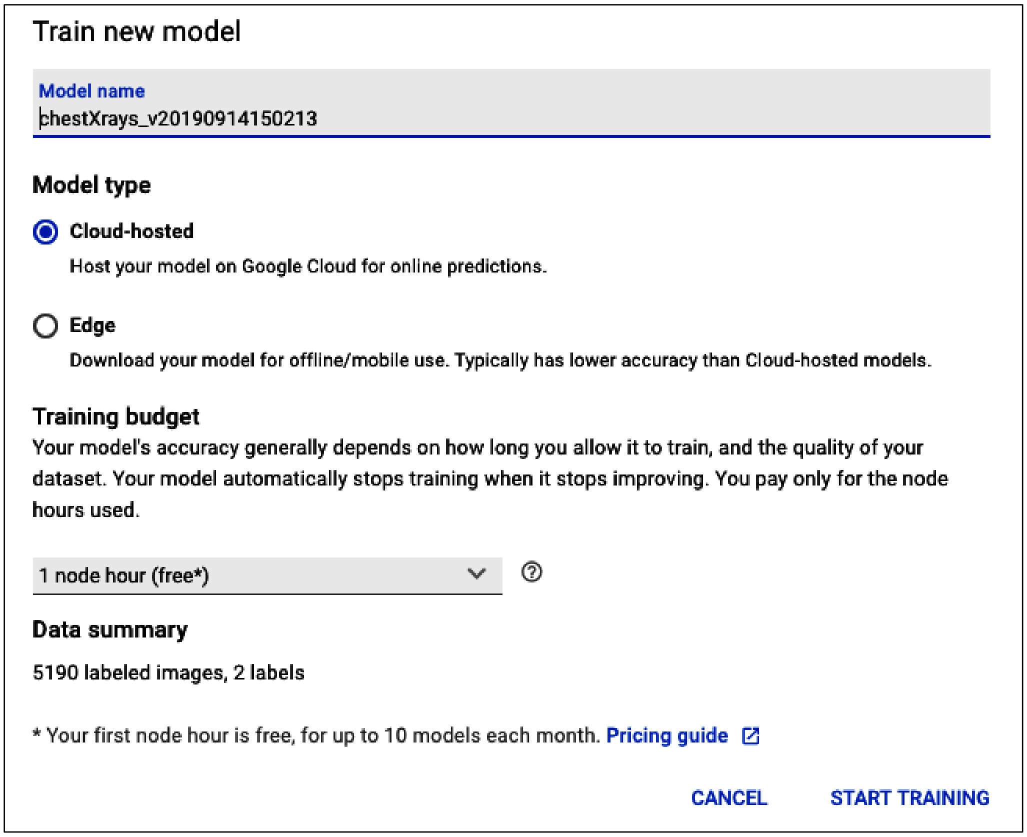


You have enough images to start training

At least **100 images** are currently assigned to each label. Learn more

Your images will be automatically split into training and test sets , so you can evaluate your model's performance. Unlabeled images will not be used.

Training images	4160
Validation images	544
Test images	486



Train new model

Model name
chestXrays_v20190914150213

Model type

● **Cloud-hosted**
Host your model on Google Cloud for online predictions.

○ **Edge**
Download your model for offline/mobile use. Typically has lower accuracy than Cloud-hosted models.

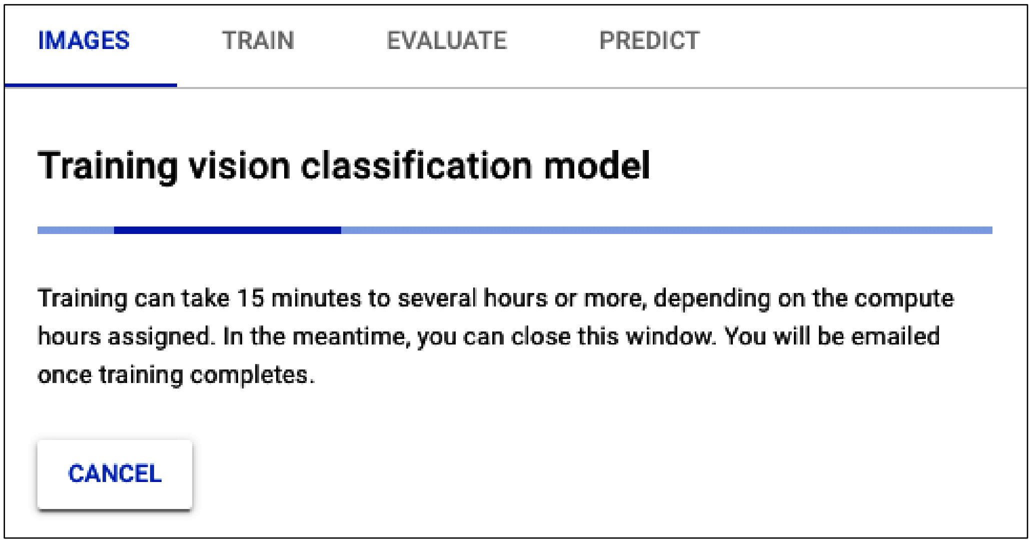
Training budget
Your model's accuracy generally depends on how long you allow it to train, and the quality of your dataset. Your model automatically stops training when it stops improving. You pay only for the node hours used.

1 node hour (free*)

Data summary

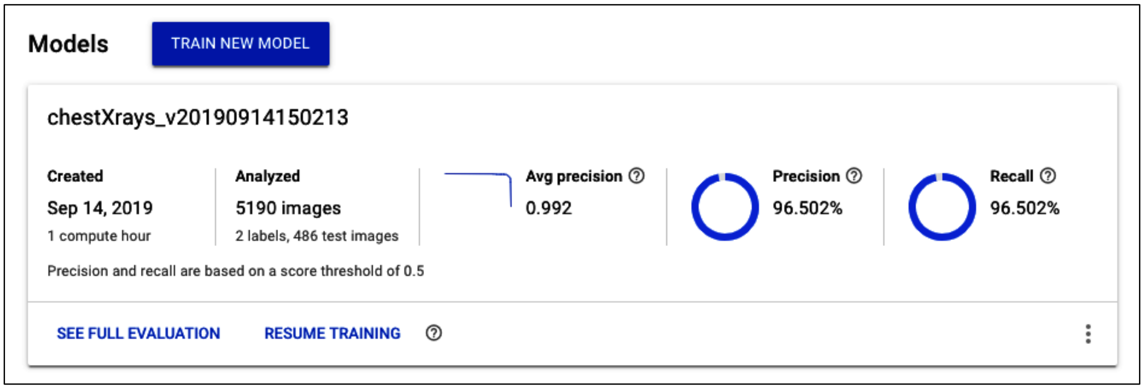
5190 labeled images, 2 labels

* Your first node hour is free, for up to 10 models each month. Pricing guide



Training vision classification model

Training can take 15 minutes to several hours or more, depending on the compute hours assigned. In the meantime, you can close this window. You will be emailed once training completes.



chestXrays_v20190914150213

Created
Sep 14, 2019
1 compute hour

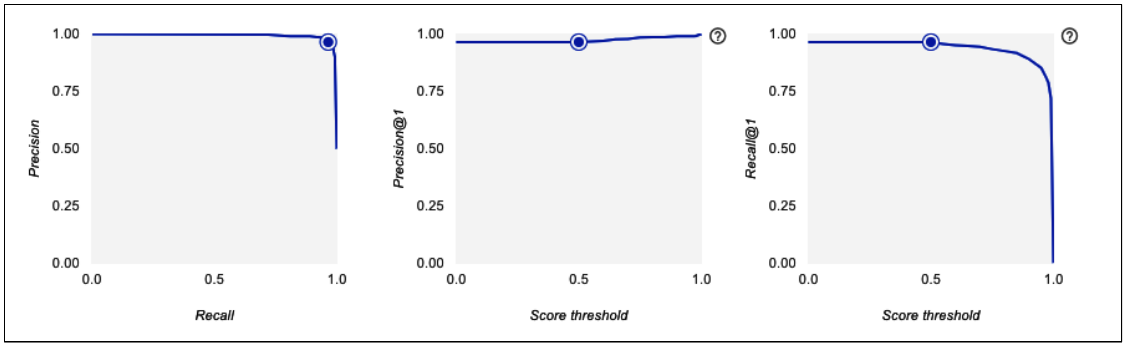
Analyzed
5190 images
2 labels, 486 test images

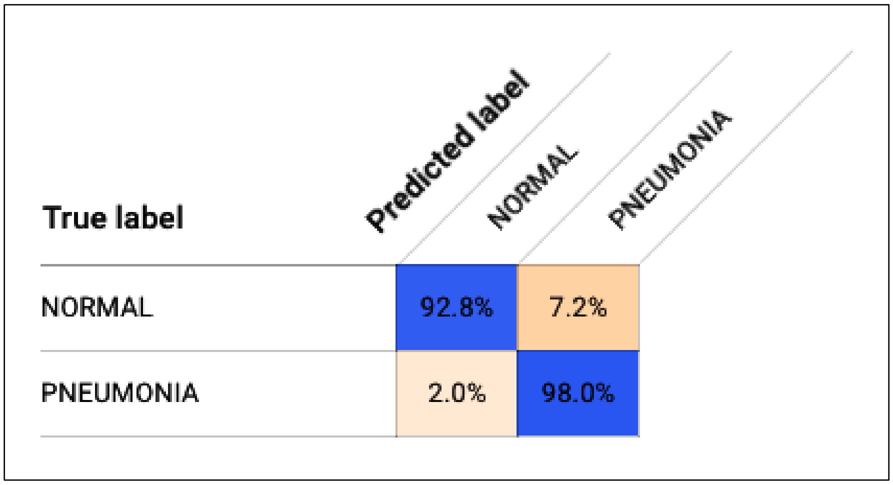
Avg precision
0.992

Precision
96.502%

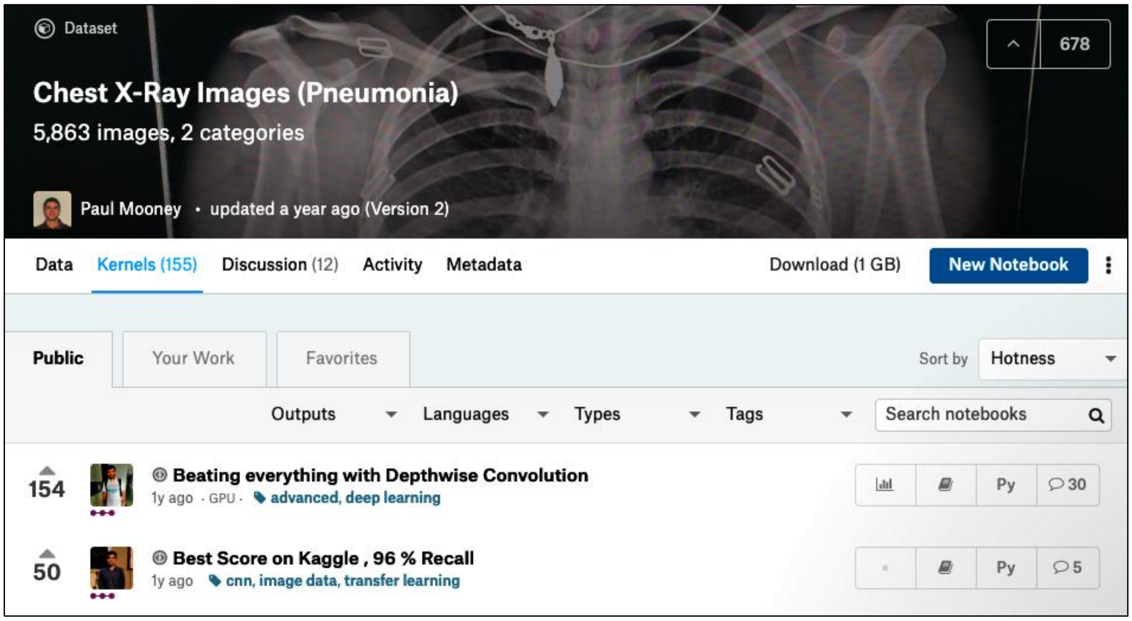
Recall
96.502%

Precision and recall are based on a score threshold of 0.5





True label	Predicted label	
	NORMAL	PNEUMONIA
NORMAL	92.8%	7.2%
PNEUMONIA	2.0%	98.0%

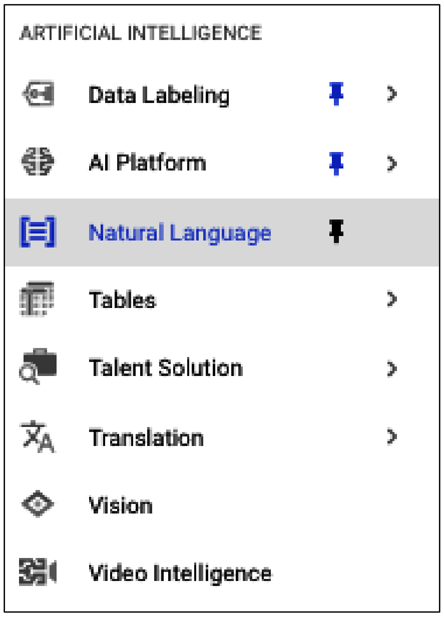


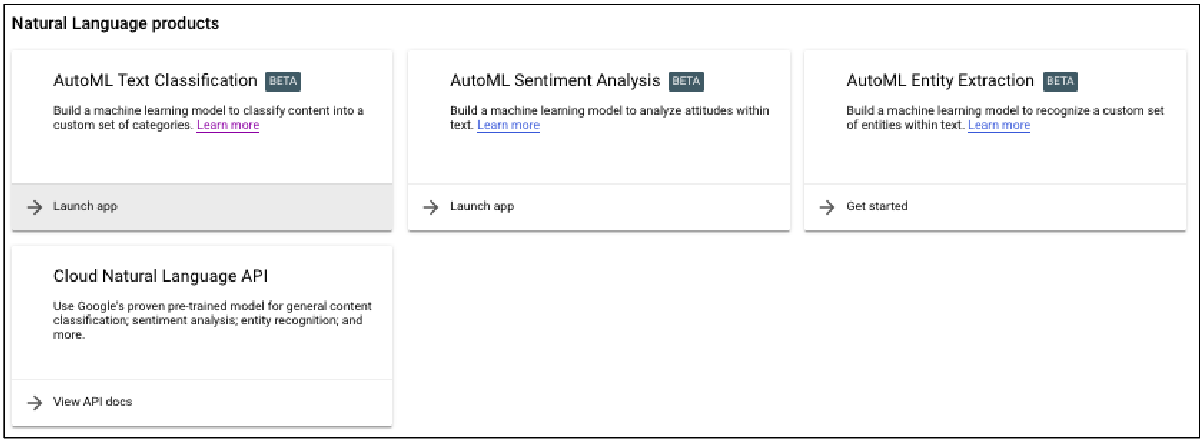
Chest X-Ray Images (Pneumonia)

5,863 images, 2 categories

Paul Mooney • updated a year ago (Version 2)

154	Beating everything with Depthwise Convolution 1y ago · GPU · advanced, deep learning	30
50	Best Score on Kaggle , 96 % Recall 1y ago cnn, image data, transfer learning	5





Natural Language products

AutoML Text Classification BETA

Build a machine learning model to classify content into a custom set of categories. Learn more

→ Launch app

AutoML Sentiment Analysis BETA

Build a machine learning model to analyze attitudes within text. Learn more

→ Launch app

AutoML Entity Extraction BETA

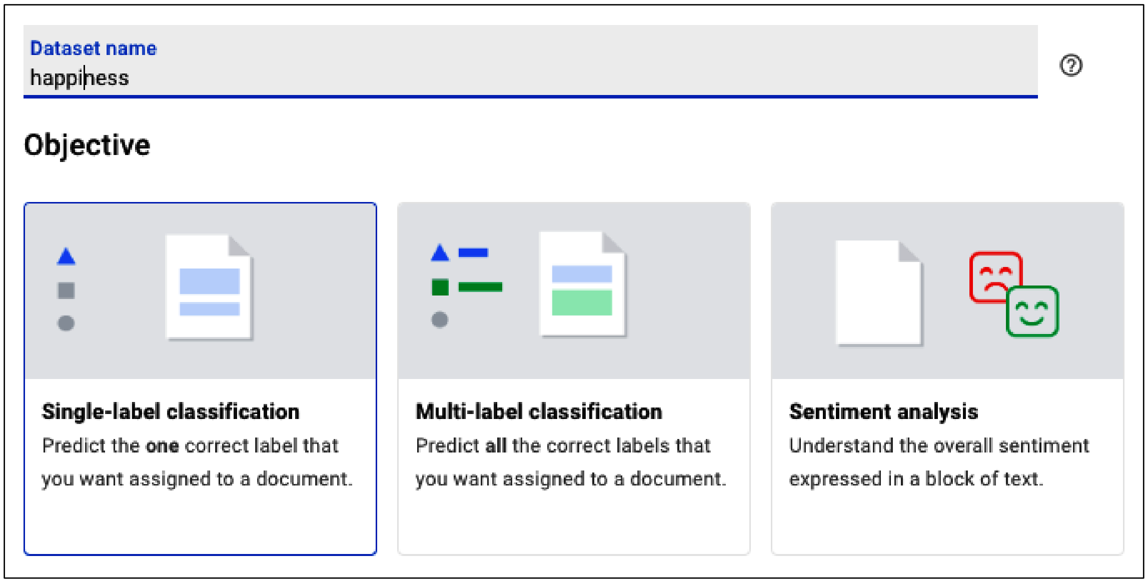
Build a machine learning model to recognize a custom set of entities within text. Learn more

→ Get started

Cloud Natural Language API

Use Google's proven pre-trained model for general content classification; sentiment analysis; entity recognition; and more.

→ View API docs



Dataset name

happiness

Objective

Single-label classification

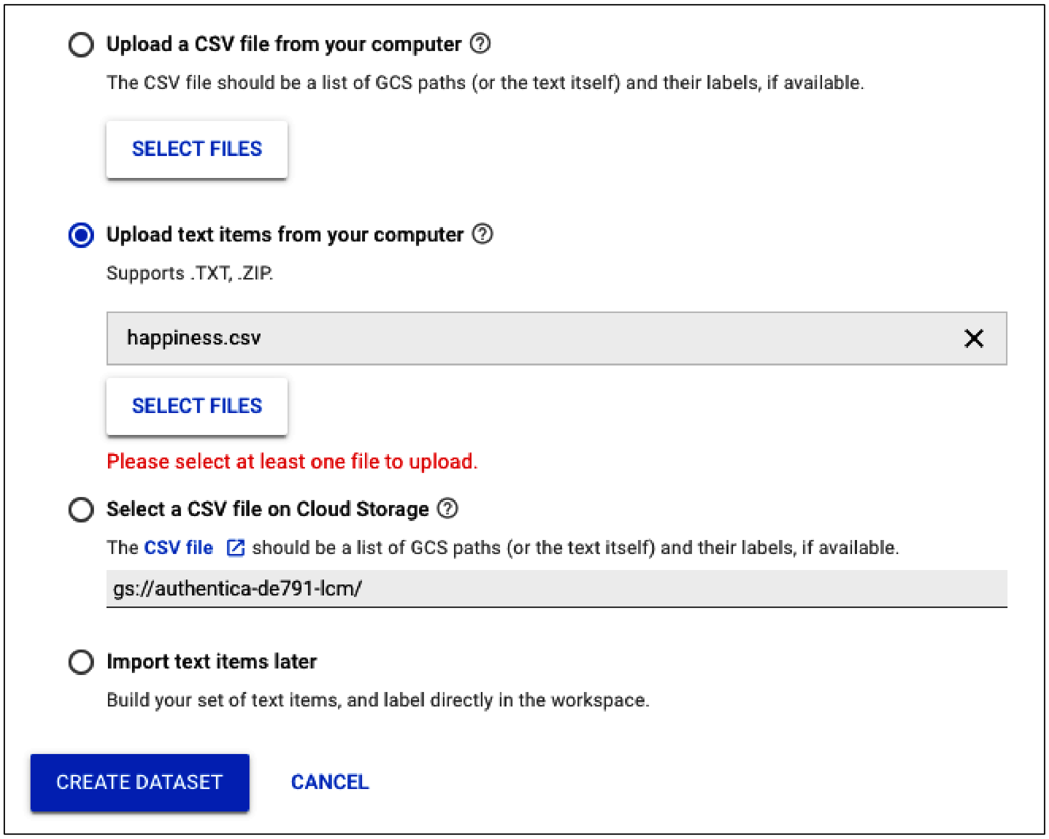
Predict the **one** correct label that you want assigned to a document.

Multi-label classification

Predict **all** the correct labels that you want assigned to a document.

Sentiment analysis

Understand the overall sentiment expressed in a block of text.



○ **Upload a CSV file from your computer** ⓘ

The CSV file should be a list of GCS paths (or the text itself) and their labels, if available.

● **Upload text items from your computer** ⓘ

Supports .TXT, .ZIP.

happiness.csv ✕

Please select at least one file to upload.

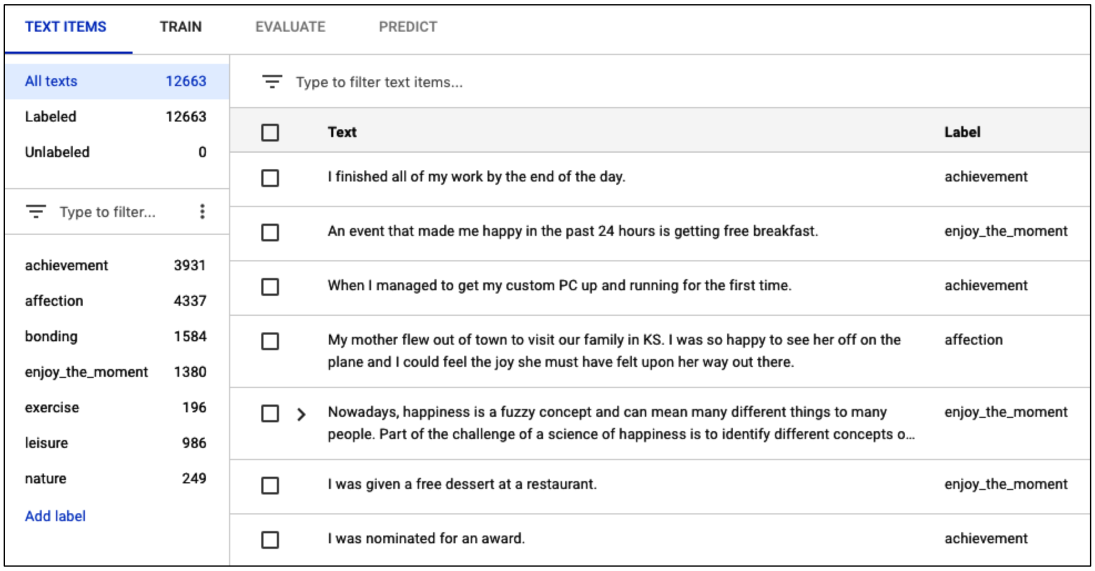
○ **Select a CSV file on Cloud Storage** ⓘ

The CSV file should be a list of GCS paths (or the text itself) and their labels, if available.

gs://authentica-de791-lcm/

○ **Import text items later**

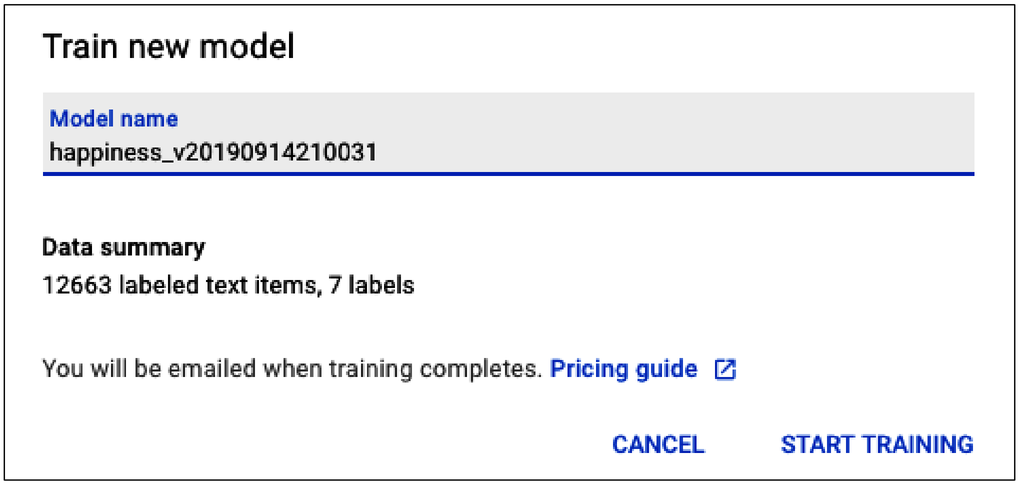
Build your set of text items, and label directly in the workspace.



All texts	12663
Labeled	12663
Unlabeled	0

achievement	3931
affection	4337
bonding	1584
enjoy_the_moment	1380
exercise	196
leisure	986
nature	249

□	Text	Label
□	I finished all of my work by the end of the day.	achievement
□	An event that made me happy in the past 24 hours is getting free breakfast.	enjoy_the_moment
□	When I managed to get my custom PC up and running for the first time.	achievement
□	My mother flew out of town to visit our family in KS. I was so happy to see her off on the plane and I could feel the joy she must have felt upon her way out there.	affection
□ ›	Nowadays, happiness is a fuzzy concept and can mean many different things to many people. Part of the challenge of a science of happiness is to identify different concepts o...	enjoy_the_moment
□	I was given a free dessert at a restaurant.	enjoy_the_moment
□	I was nominated for an award.	achievement

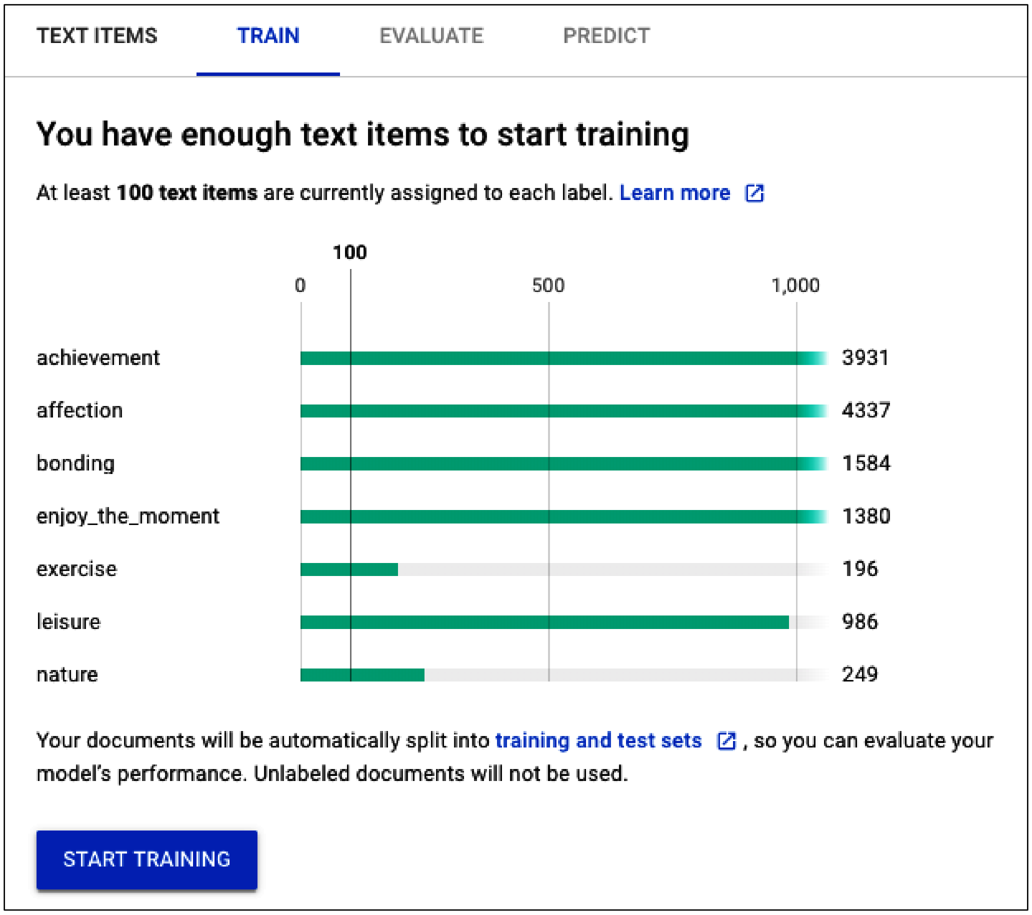


Train new model

Model name
happiness_v20190914210031

Data summary
12663 labeled text items, 7 labels

You will be emailed when training completes. Pricing guide

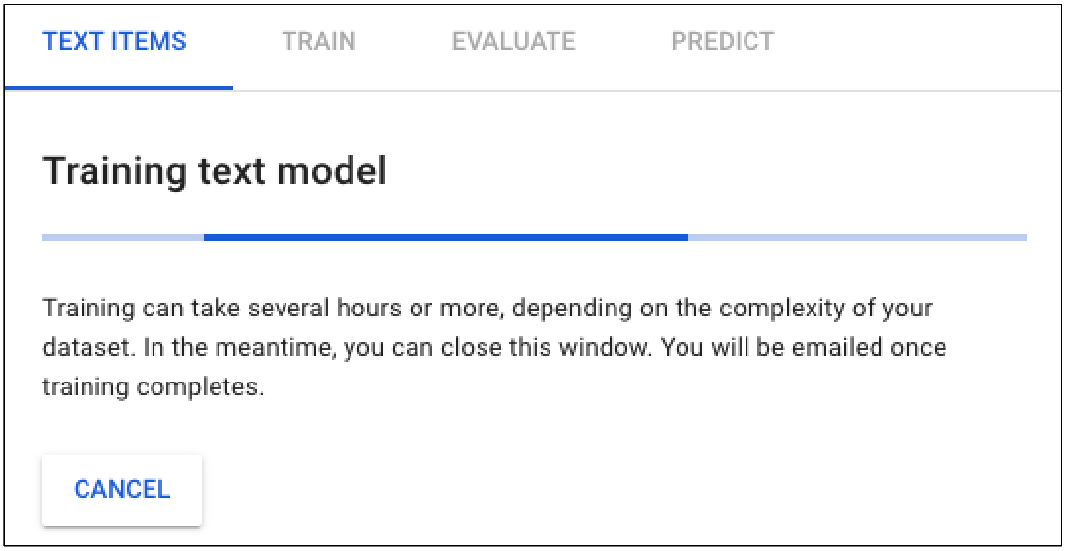


You have enough text items to start training

At least **100 text items** are currently assigned to each label. Learn more

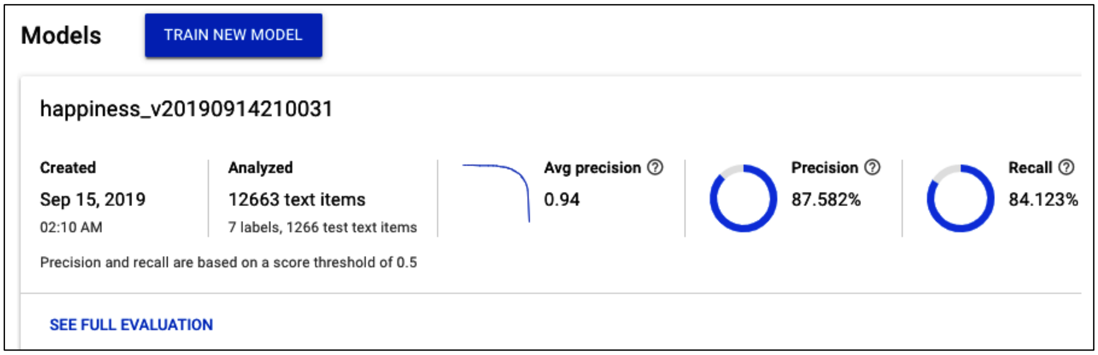
achievement	3931
affection	4337
bonding	1584
enjoy_the_moment	1380
exercise	196
leisure	986
nature	249

Your documents will be automatically split into training and test sets , so you can evaluate your model's performance. Unlabeled documents will not be used.



Training text model

Training can take several hours or more, depending on the complexity of your dataset. In the meantime, you can close this window. You will be emailed once training completes.



happiness_v20190914210031

Created
Sep 15, 2019
02:10 AM

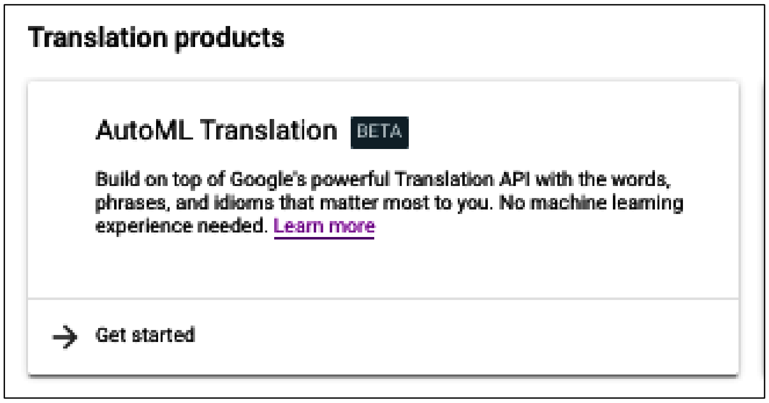
Analyzed
12663 text items
7 labels, 1266 test text items

Avg precision ⓘ
0.94

Precision ⓘ
87.582%

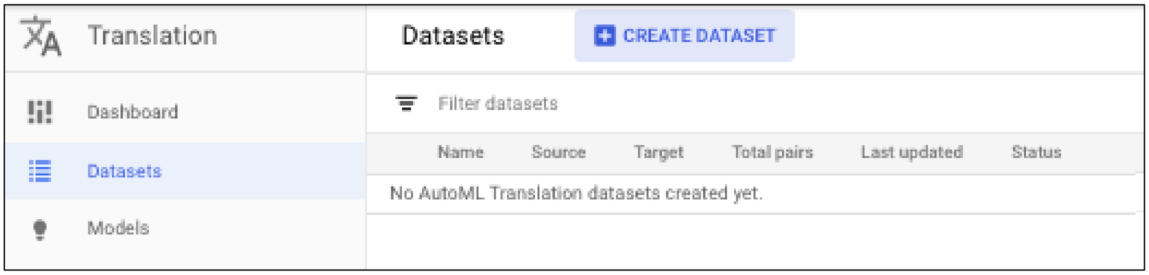
Recall ⓘ
84.123%

Precision and recall are based on a score threshold of 0.5



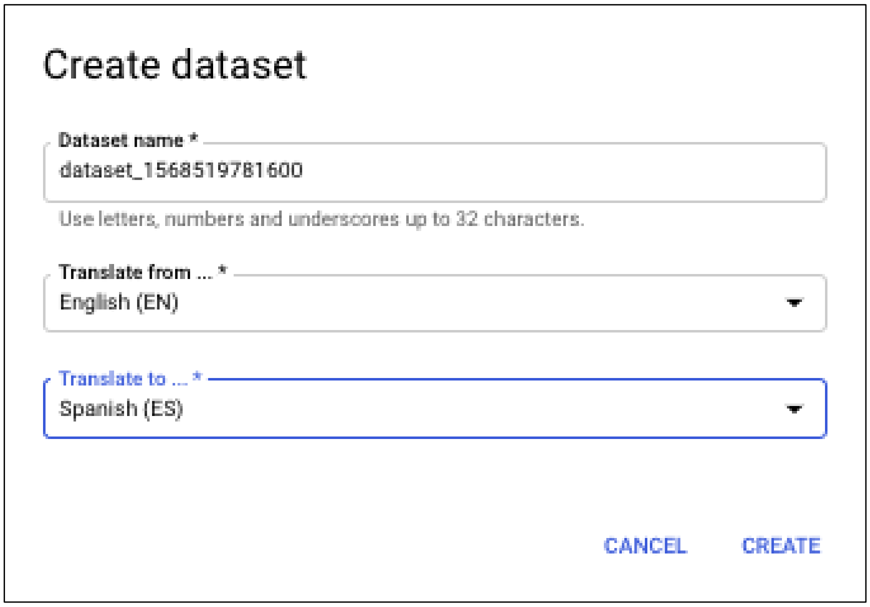
AutoML Translation

Build on top of Google's powerful Translation API with the words, phrases, and idioms that matter most to you. No machine learning experience needed. Learn more



Name	Source	Target	Total pairs	Last updated	Status

No AutoML Translation datasets created yet.

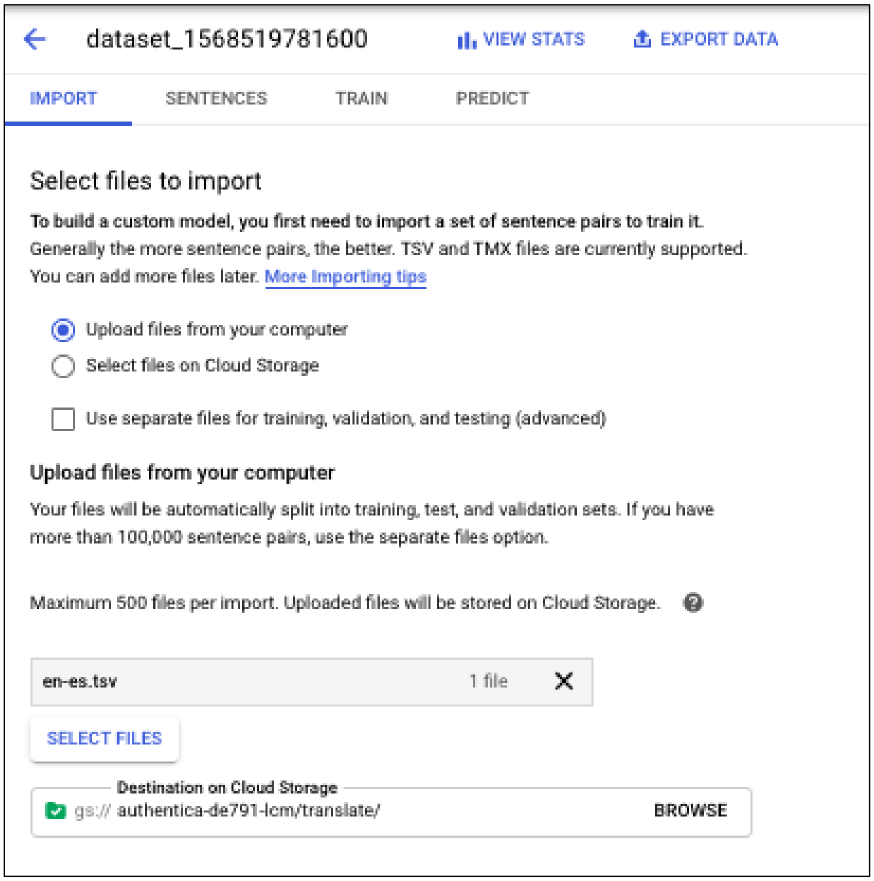


Create dataset

Dataset name *
dataset_1568519781600
Use letters, numbers and underscores up to 32 characters.

Translate from ... *
English (EN)

Translate to ... *
Spanish (ES)



Select files to import

To build a custom model, you first need to import a set of sentence pairs to train it. Generally the more sentence pairs, the better. TSV and TMX files are currently supported. You can add more files later. More Importing tips

● Upload files from your computer
○ Select files on Cloud Storage

□ Use separate files for training, validation, and testing (advanced)

Upload files from your computer

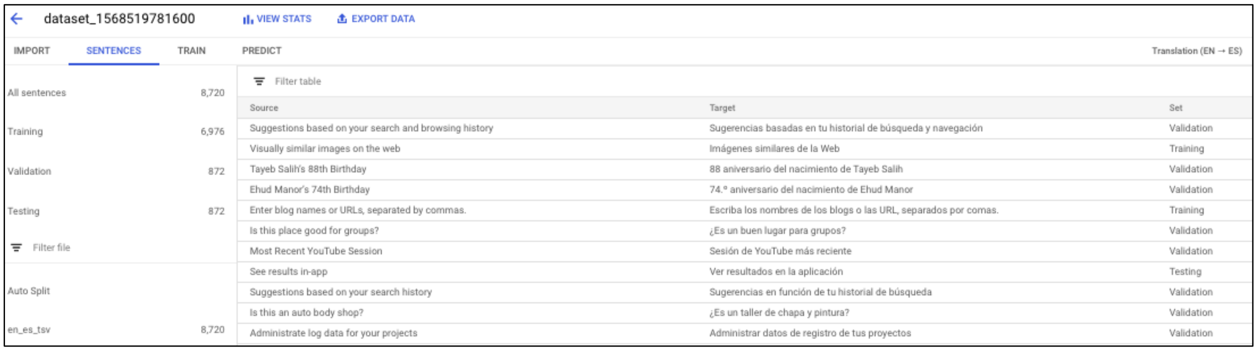
Your files will be automatically split into training, test, and validation sets. If you have more than 100,000 sentence pairs, use the separate files option.

Maximum 500 files per import. Uploaded files will be stored on Cloud Storage.

en-es.tsv 1 file

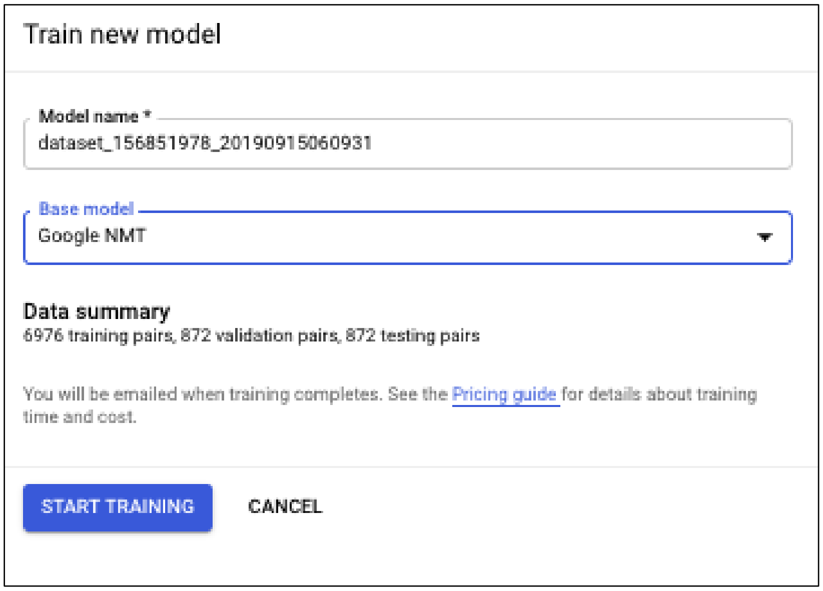
Destination on Cloud Storage

gs:// authentica-de791-lcm/translate/



All sentences	8,720
Training	6,976
Validation	872
Testing	872
Auto Split	
en_es_tsv	8,720

Source	Target	Set
Suggestions based on your search and browsing history	Sugerencias basadas en tu historial de búsqueda y navegación	Validation
Visually similar images on the web	Imágenes similares de la Web	Training
Tayeb Salih's 88th Birthday	88 aniversario del nacimiento de Tayeb Salih	Validation
Ehud Manor's 74th Birthday	74.º aniversario del nacimiento de Ehud Manor	Validation
Enter blog names or URLs, separated by commas.	Escriba los nombres de los blogs o las URL, separados por comas.	Training
Is this place good for groups?	¿Es un buen lugar para grupos?	Validation
Most Recent YouTube Session	Sesión de YouTube más reciente	Validation
See results in-app	Ver resultados en la aplicación	Testing
Suggestions based on your search history	Sugerencias en función de tu historial de búsqueda	Validation
Is this an auto body shop?	¿Es un taller de chapa y pintura?	Validation
Administrate log data for your projects	Administrar datos de registro de tus proyectos	Validation



Train new model

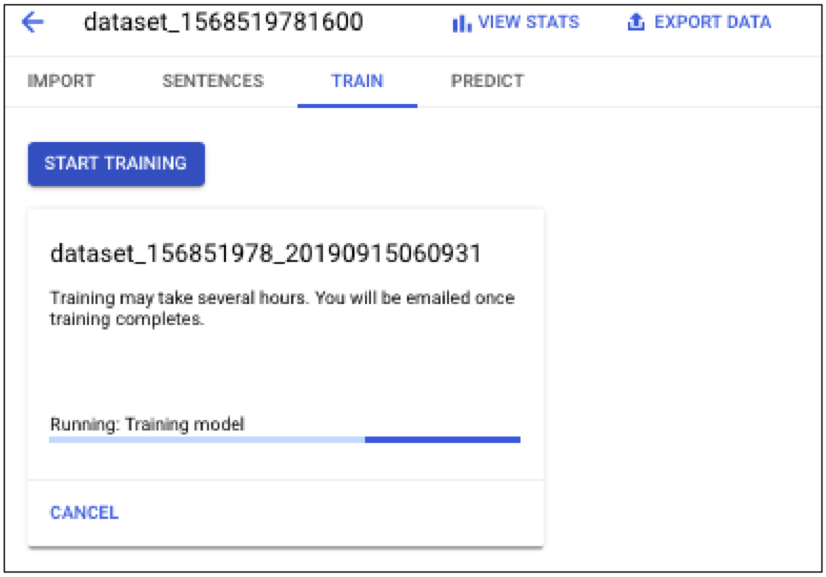
Model name *: dataset_156851978_20190915060931

Base model: Google NMT

Data summary

6976 training pairs, 872 validation pairs, 872 testing pairs

You will be emailed when training completes. See the Pricing guide for details about training time and cost.

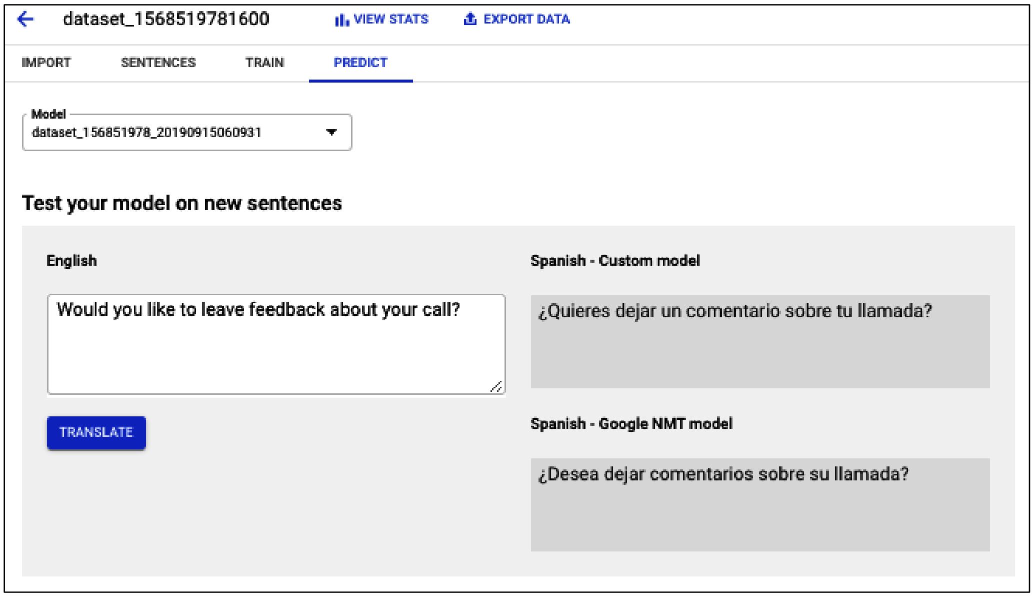


dataset_1568519781600

dataset_156851978_20190915060931

Training may take several hours. You will be emailed once training completes.

Running: Training model



Test your model on new sentences

English

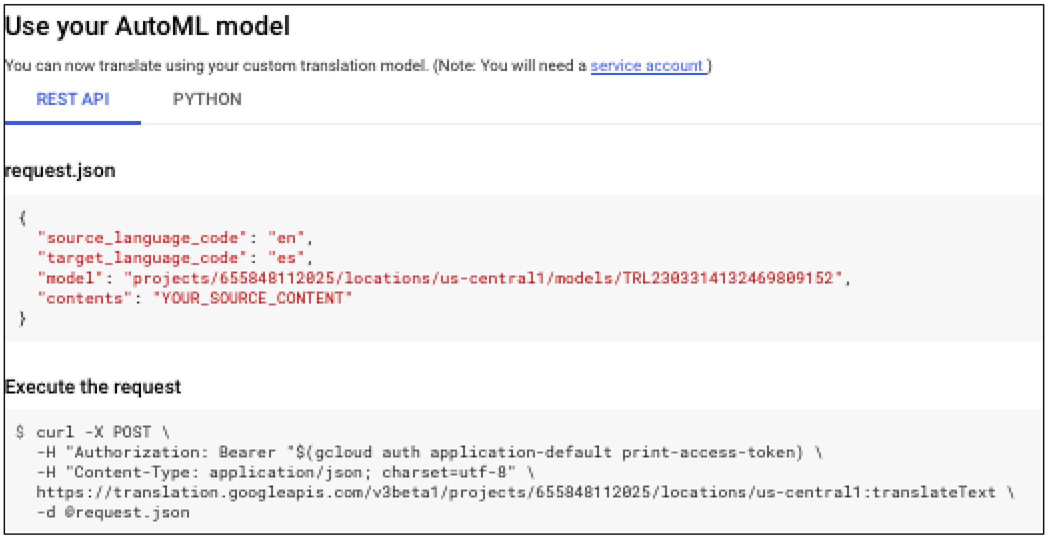
Would you like to leave feedback about your call?

Spanish - Custom model

¿Quieres dejar un comentario sobre tu llamada?

Spanish - Google NMT model

¿Desea dejar comentarios sobre su llamada?



Use your AutoML model

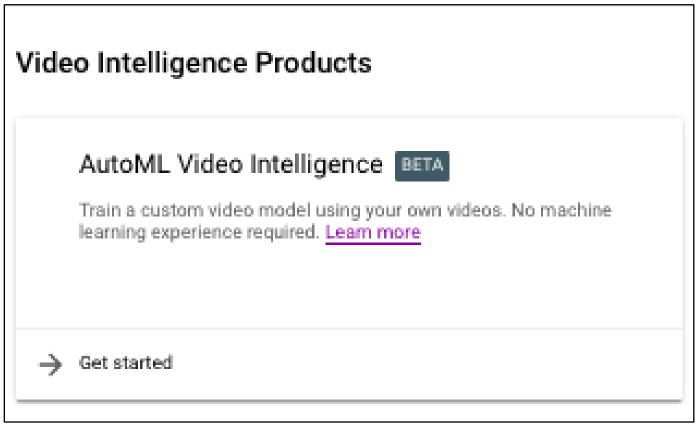
You can now translate using your custom translation model. (Note: You will need a service account)

request.json

```
{
  "source_language_code": "en",
  "target_language_code": "es",
  "model": "projects/655848112025/locations/us-central1/models/TRL2303314132469809152",
  "contents": "YOUR_SOURCE_CONTENT"
}
```

Execute the request

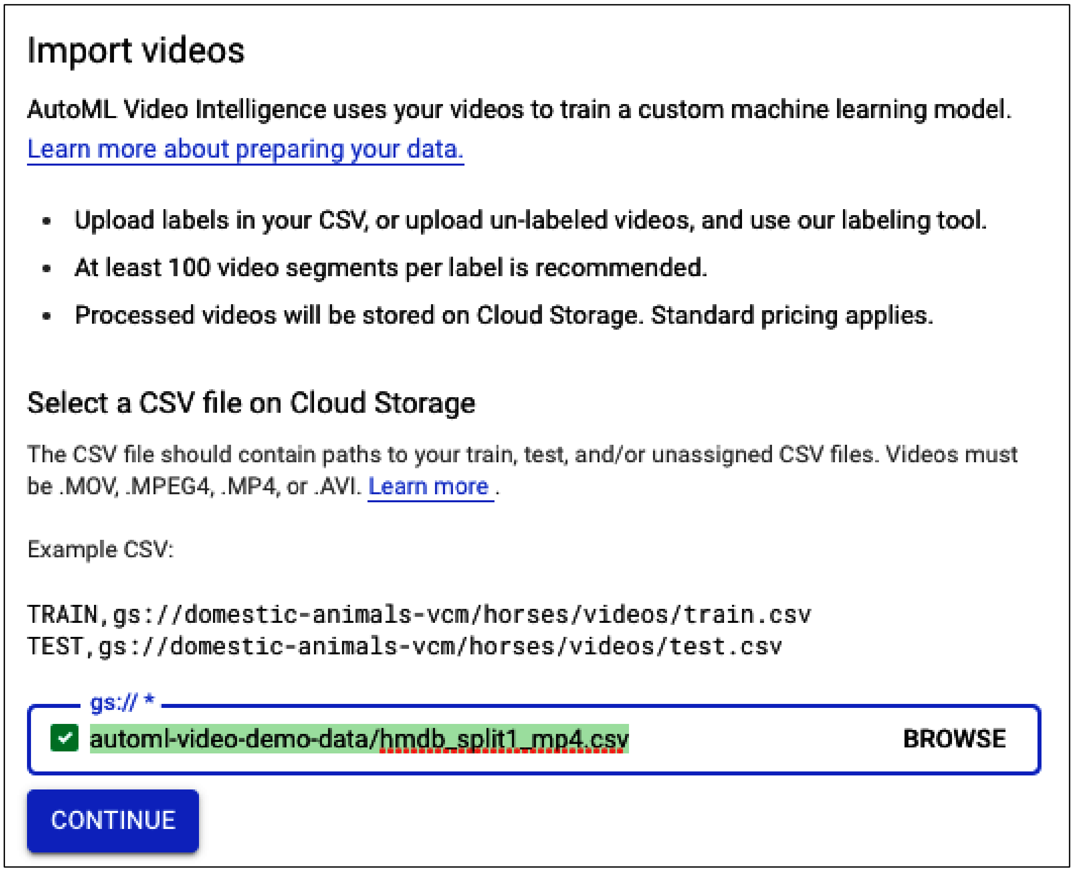
```
$ curl -X POST \
  -H "Authorization: Bearer "$(gcloud auth application-default print-access-token) \
  -H "Content-Type: application/json; charset=utf-8" \
  https://translation.googleapis.com/v3beta1/projects/655848112025/locations/us-central1:translateText \
  -d @request.json
```



Video Intelligence Products

AutoML Video Intelligence BETA

Train a custom video model using your own videos. No machine learning experience required. Learn more



Import videos

AutoML Video Intelligence uses your videos to train a custom machine learning model. Learn more about preparing your data.

- Upload labels in your CSV, or upload un-labeled videos, and use our labeling tool.
- At least 100 video segments per label is recommended.
- Processed videos will be stored on Cloud Storage. Standard pricing applies.

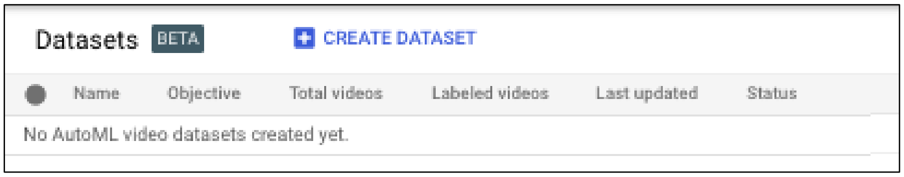
Select a CSV file on Cloud Storage

The CSV file should contain paths to your train, test, and/or unassigned CSV files. Videos must be .MOV, .MPEG4, .MP4, or .AVI. Learn more .

Example CSV:

```
TRAIN,gs://domestic-animals-vcm/horses/videos/train.csv
TEST,gs://domestic-animals-vcm/horses/videos/test.csv
```

gs:// *: ✓ automl-video-demo-data/hmdb_split1_mp4.csv

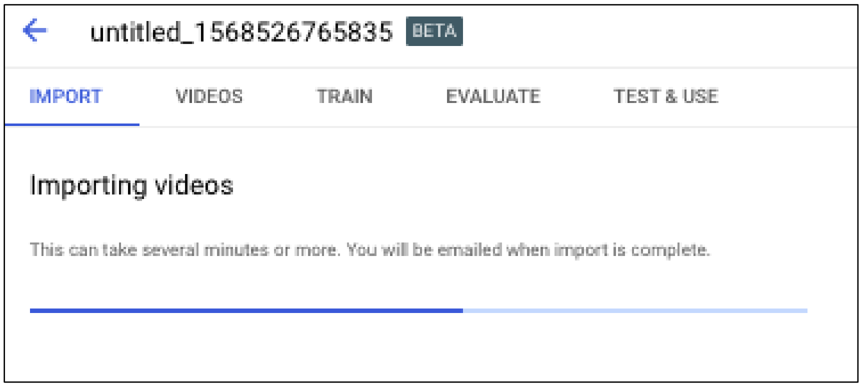


Datasets BETA

CREATE DATASET

Name	Objective	Total videos	Labeled videos	Last updated	Status

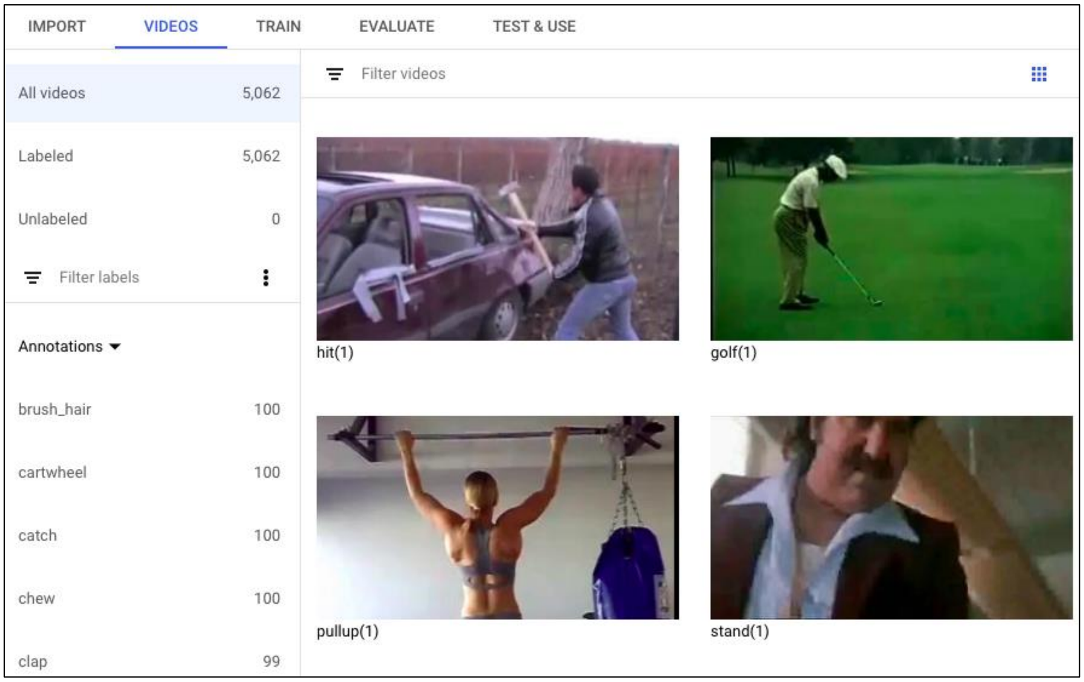
No AutoML video datasets created yet.



untitled_1568526765835 BETA

Importing videos

This can take several minutes or more. You will be emailed when import is complete.



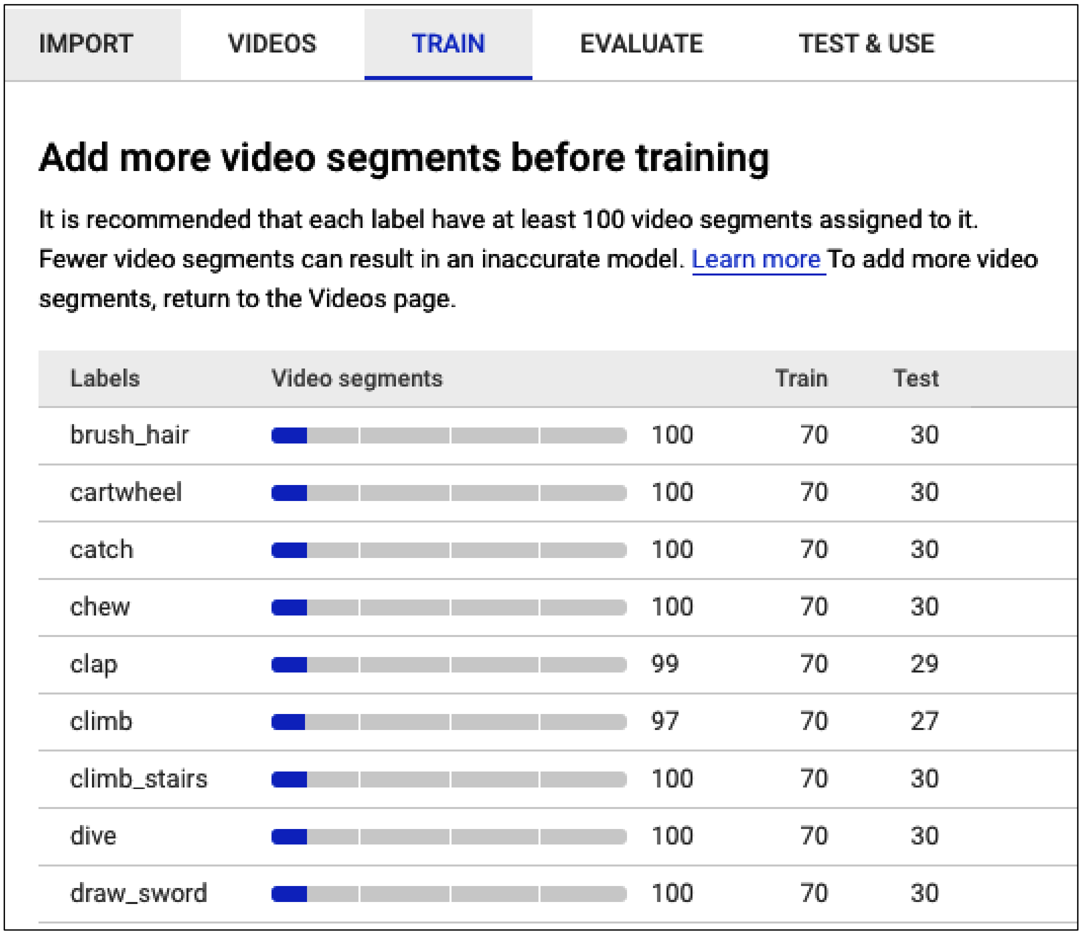
All videos	5,062
Labeled	5,062
Unlabeled	0
brush_hair	100
cartwheel	100
catch	100
chew	100
clap	99

hit(1)

golf(1)

pullup(1)

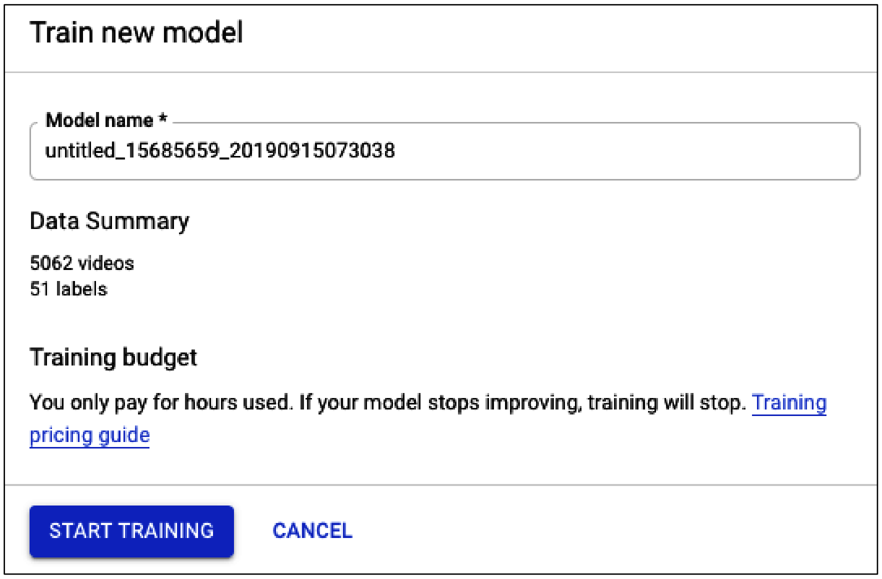
stand(1)



Add more video segments before training

It is recommended that each label have at least 100 video segments assigned to it. Fewer video segments can result in an inaccurate model. Learn more To add more video segments, return to the Videos page.

Labels	Video segments	Train	Test
brush_hair	100	70	30
cartwheel	100	70	30
catch	100	70	30
chew	100	70	30
clap	99	70	29
climb	97	70	27
climb_stairs	100	70	30
dive	100	70	30
draw_sword	100	70	30



Train new model

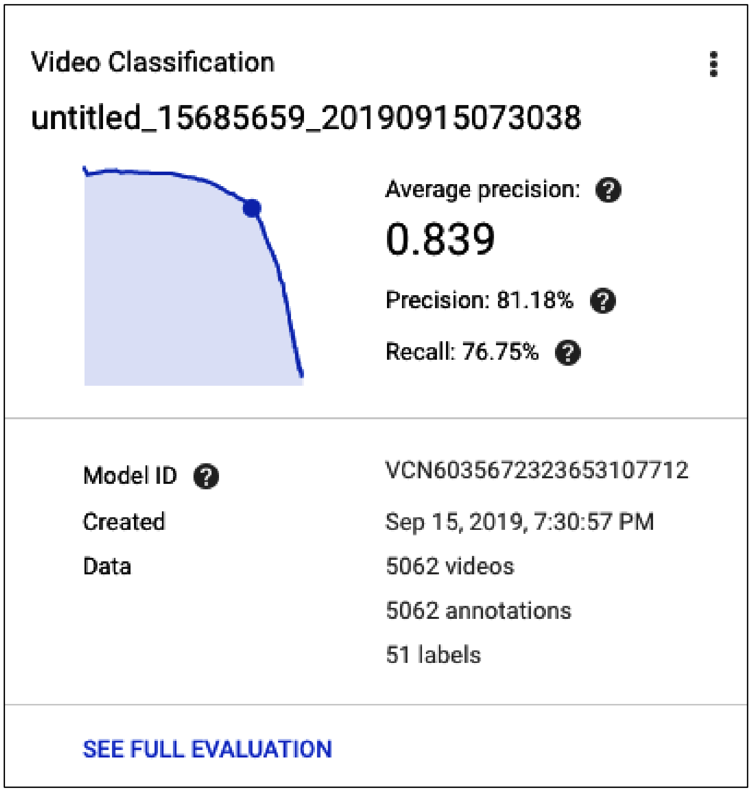
Model name *: untitled_15685659_20190915073038

Data Summary

5062 videos
51 labels

Training budget

You only pay for hours used. If your model stops improving, training will stop. Training pricing guide



Video Classification

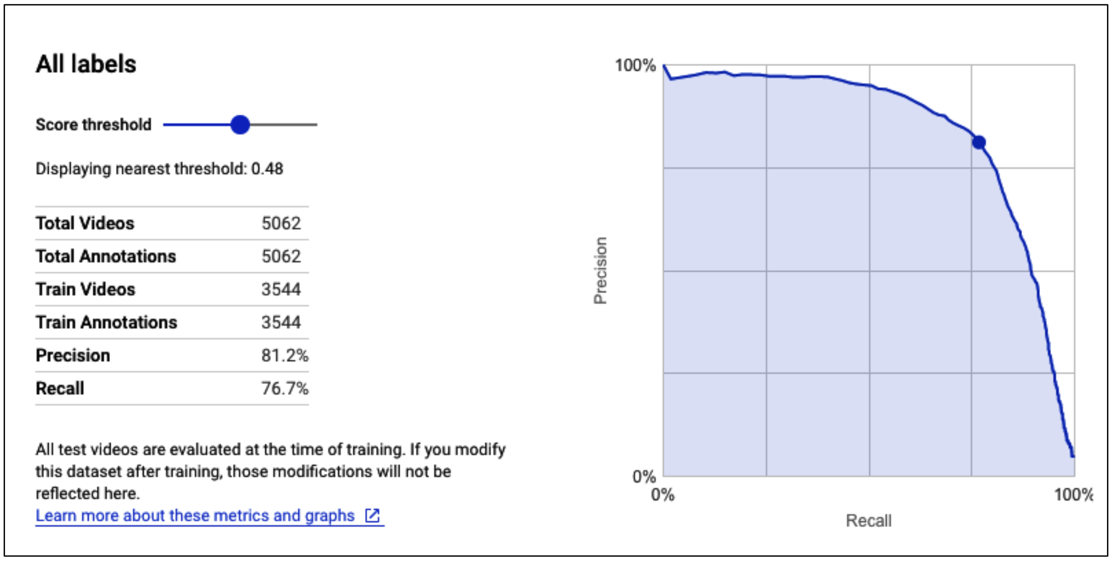
untitled_15685659_20190915073038

Average precision: 0.839

Precision: 81.18%

Recall: 76.75%

Model ID	VCN6035672323653107712
Created	Sep 15, 2019, 7:30:57 PM
Data	5062 videos 5062 annotations 51 labels



All labels

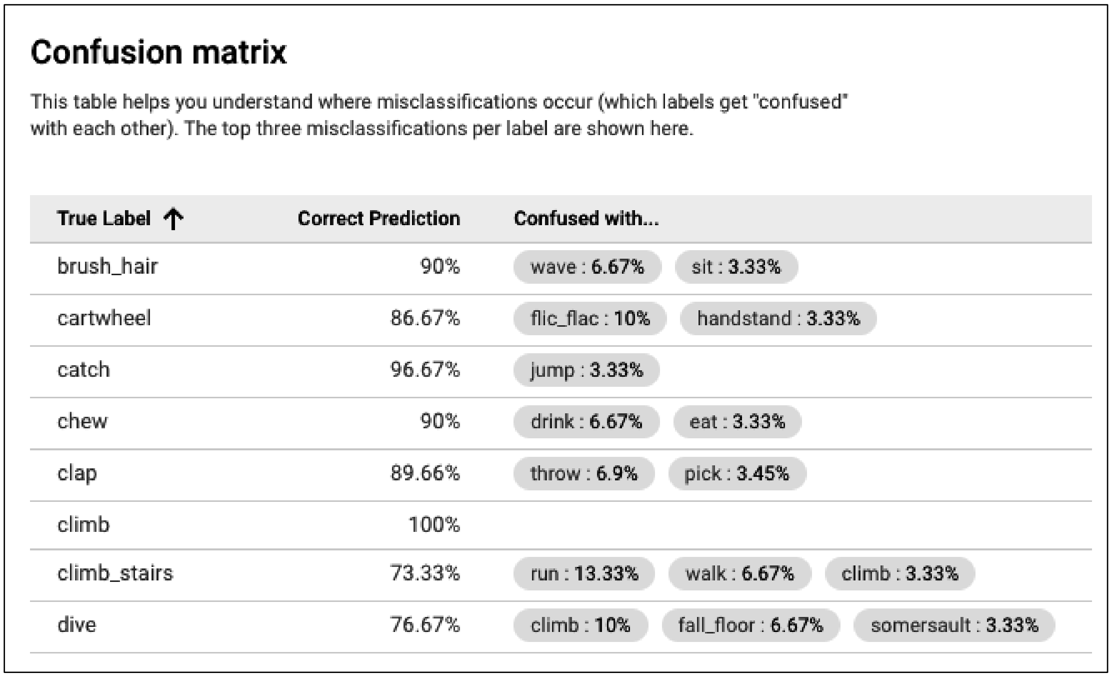
Score threshold

Displaying nearest threshold: 0.48

Total Videos	5062
Total Annotations	5062
Train Videos	3544
Train Annotations	3544
Precision	81.2%
Recall	76.7%

All test videos are evaluated at the time of training. If you modify this dataset after training, those modifications will not be reflected here.

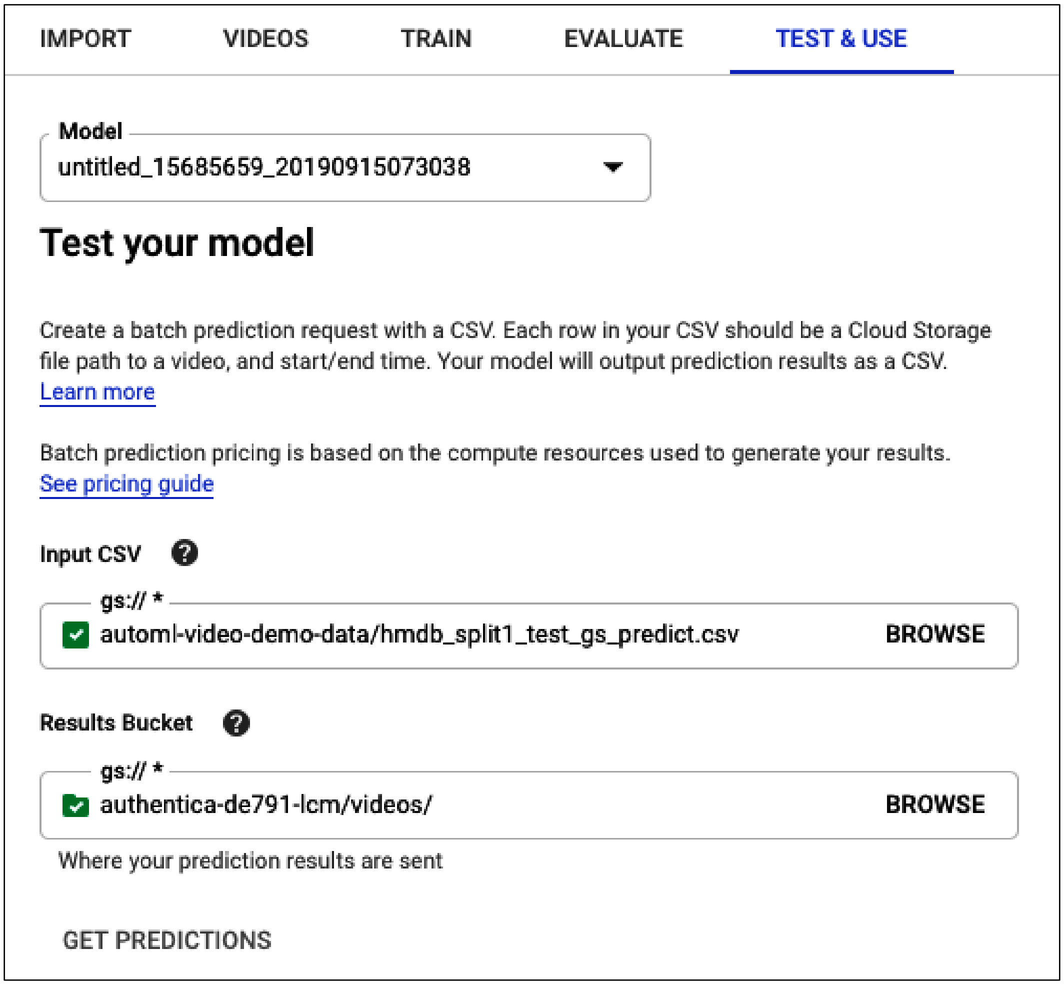
Learn more about these metrics and graphs



Confusion matrix

This table helps you understand where misclassifications occur (which labels get "confused" with each other). The top three misclassifications per label are shown here.

True Label ↑	Correct Prediction	Confused with...		
brush_hair	90%	wave : 6.67%	sit : 3.33%	
cartwheel	86.67%	flic_flac : 10%	handstand : 3.33%	
catch	96.67%	jump : 3.33%		
chew	90%	drink : 6.67%	eat : 3.33%	
clap	89.66%	throw : 6.9%	pick : 3.45%	
climb	100%			
climb_stairs	73.33%	run : 13.33%	walk : 6.67%	climb : 3.33%
dive	76.67%	climb : 10%	fall_floor : 6.67%	somersault : 3.33%



Model
untitled_15685659_20190915073038

Test your model

Create a batch prediction request with a CSV. Each row in your CSV should be a Cloud Storage file path to a video, and start/end time. Your model will output prediction results as a CSV.
Learn more

Batch prediction pricing is based on the compute resources used to generate your results.
See pricing guide

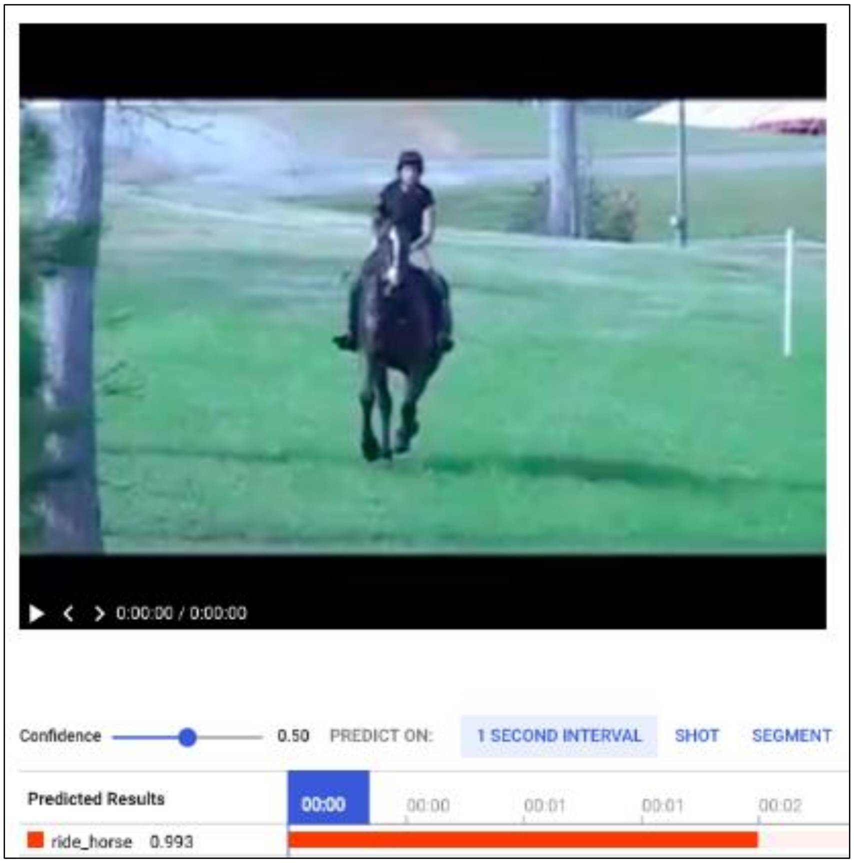
Input CSV

gs:// *
✓ automl-video-demo-data/hmdb_split1_test_gs_predict.csv

Results Bucket

gs:// *
✓ authentica-de791-lcm/videos/

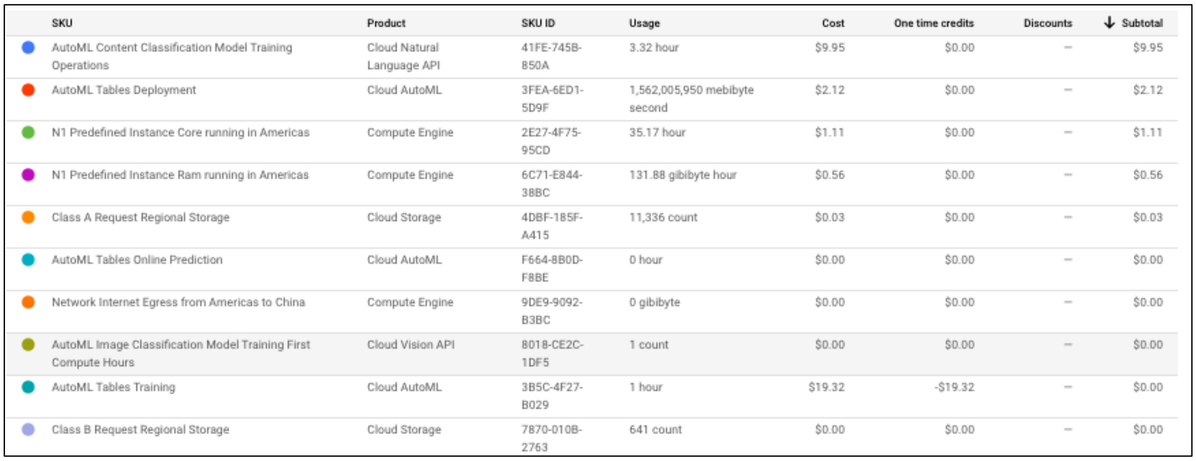
Where your prediction results are sent



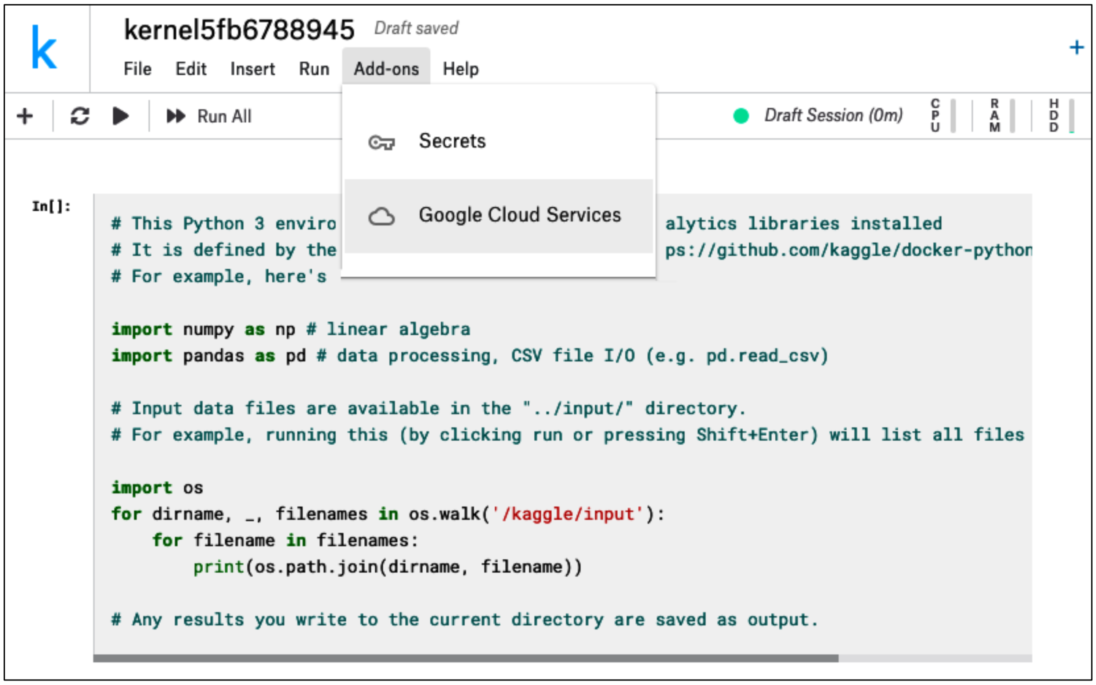
Confidence 0.50 PREDICT ON: 1 SECOND INTERVAL SHOT SEGMENT

Predicted Results

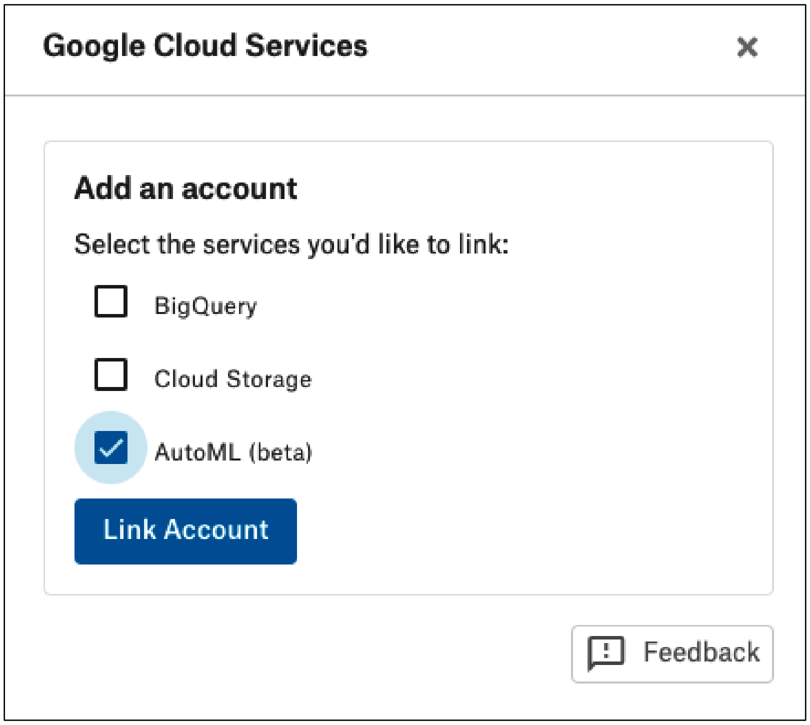
🟥 ride_horse 0.993



	SKU	Product	SKU ID	Usage	Cost	One time credits	Discounts	↓ Subtotal
●	AutoML Content Classification Model Training Operations	Cloud Natural Language API	41FE-745B-850A	3.32 hour	$9.95	$0.00	–	$9.95
●	AutoML Tables Deployment	Cloud AutoML	3FEA-6ED1-5D9F	1,562,005,950 mebibyte second	$2.12	$0.00	–	$2.12
●	N1 Predefined Instance Core running in Americas	Compute Engine	2E27-4F75-95CD	35.17 hour	$1.11	$0.00	–	$1.11
●	N1 Predefined Instance Ram running in Americas	Compute Engine	6C71-E844-38BC	131.88 gibibyte hour	$0.56	$0.00	–	$0.56
●	Class A Request Regional Storage	Cloud Storage	4DBF-185F-A415	11,336 count	$0.03	$0.00	–	$0.03
●	AutoML Tables Online Prediction	Cloud AutoML	F664-8B0D-F8BE	0 hour	$0.00	$0.00	–	$0.00
●	Network Internet Egress from Americas to China	Compute Engine	9DE9-9092-B3BC	0 gibibyte	$0.00	$0.00	–	$0.00
●	AutoML Image Classification Model Training First Compute Hours	Cloud Vision API	8018-CE2C-1DF5	1 count	$0.00	$0.00	–	$0.00
●	AutoML Tables Training	Cloud AutoML	3B5C-4F27-B029	1 hour	$19.32	-$19.32	–	$0.00
●	Class B Request Regional Storage	Cloud Storage	7870-010B-2763	641 count	$0.00	$0.00	–	$0.00



```
# This Python 3 enviro                                alytics libraries installed
# It is defined by the                                ps://github.com/kaggle/docker-python
# For example, here's

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list all files

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# Any results you write to the current directory are saved as output.
```



Google Cloud Services ✕

Add an account

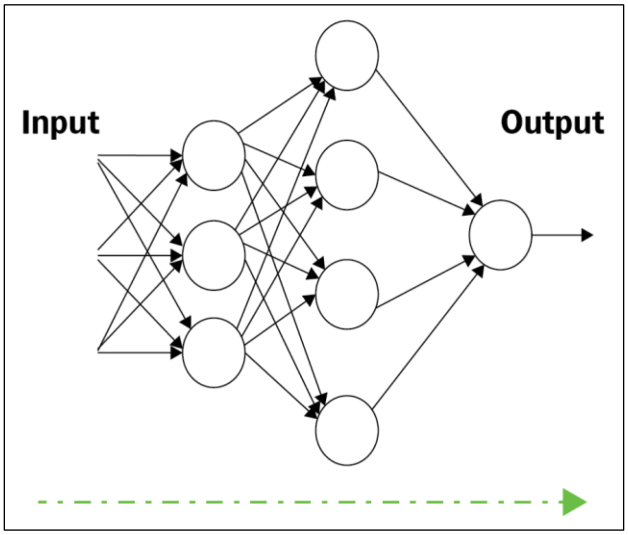
Select the services you'd like to link:

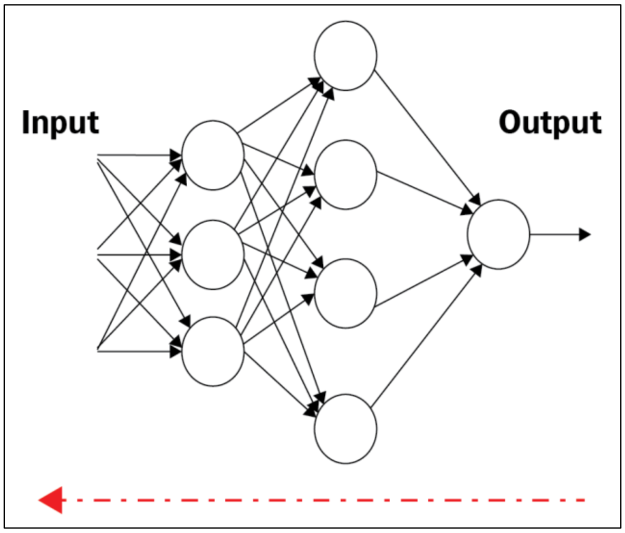
☐ BigQuery

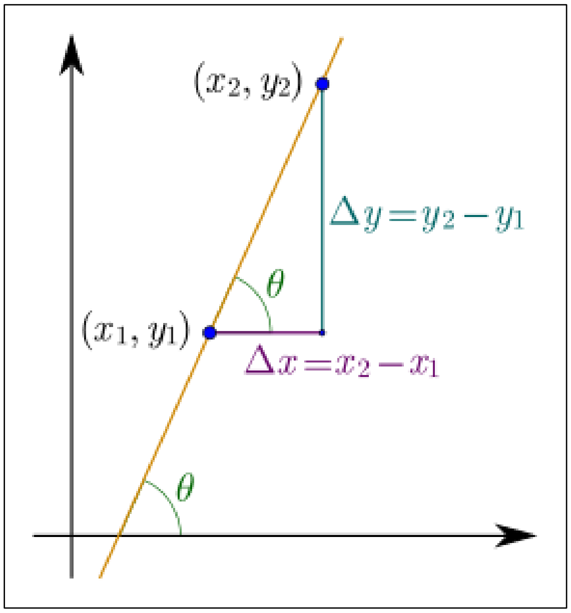
☐ Cloud Storage

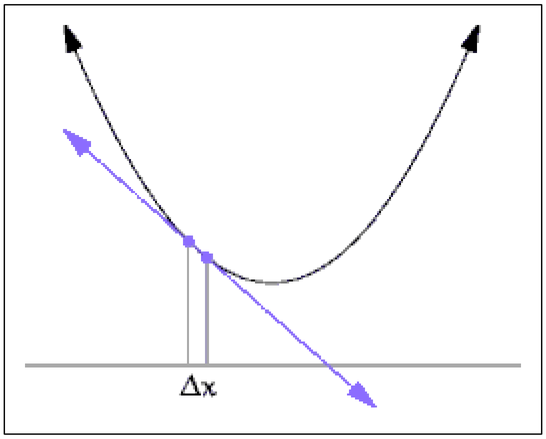
☑ AutoML (beta)

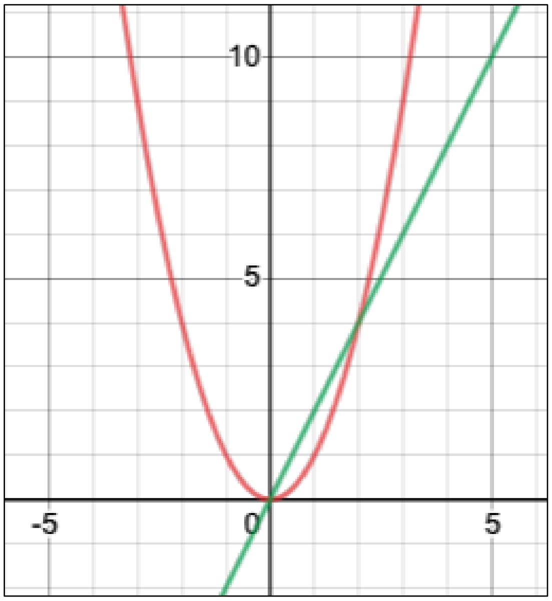
Chapter 15: The Math Behind Deep Learning

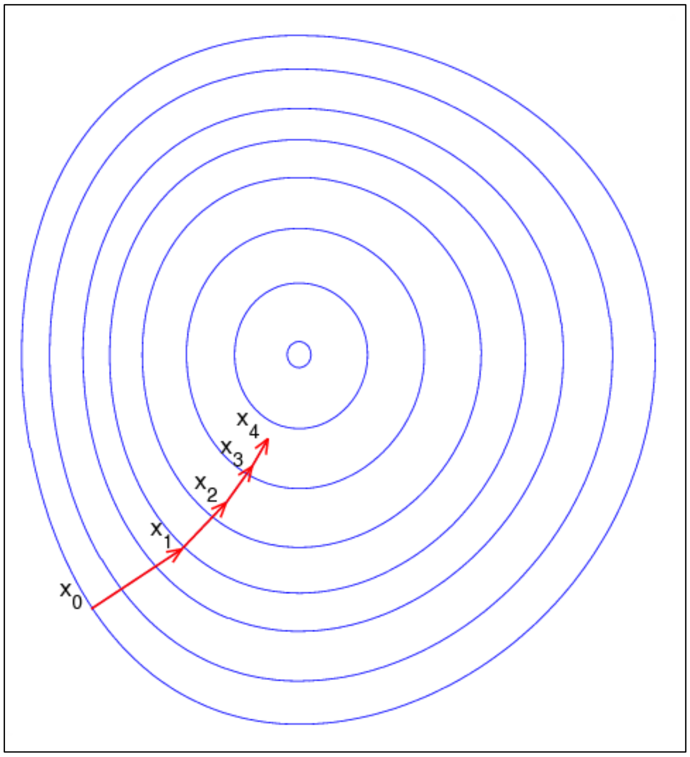


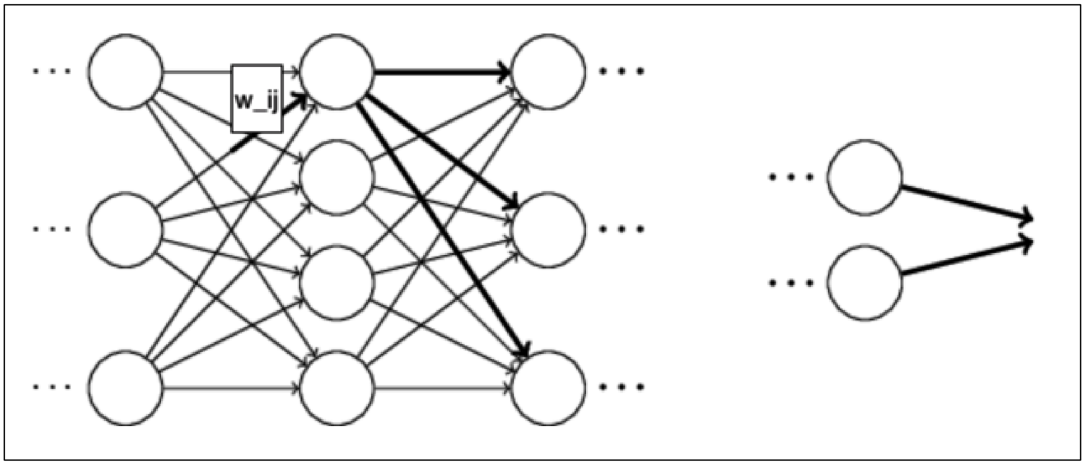


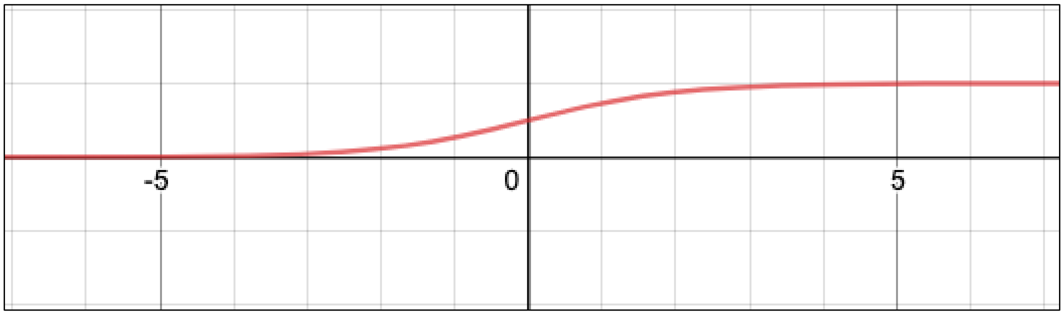

(x_2, y_2)
$\Delta y = y_2 - y_1$
θ
(x_1, y_1)
$\Delta x = x_2 - x_1$
θ

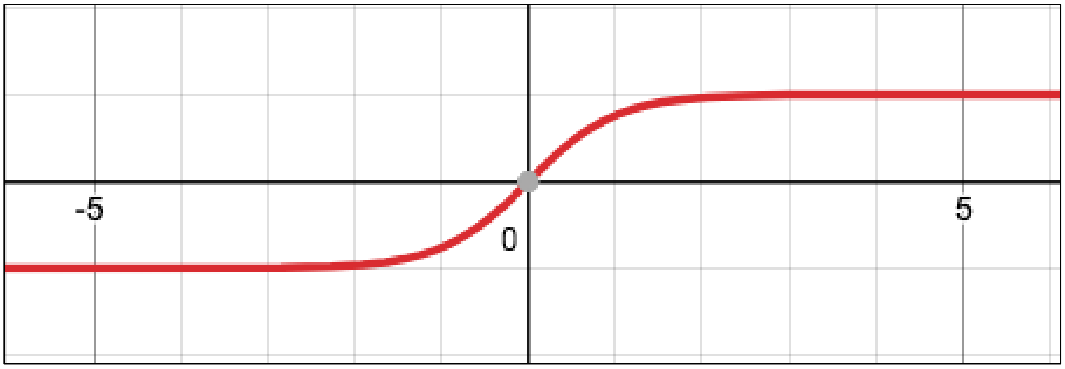

Δx

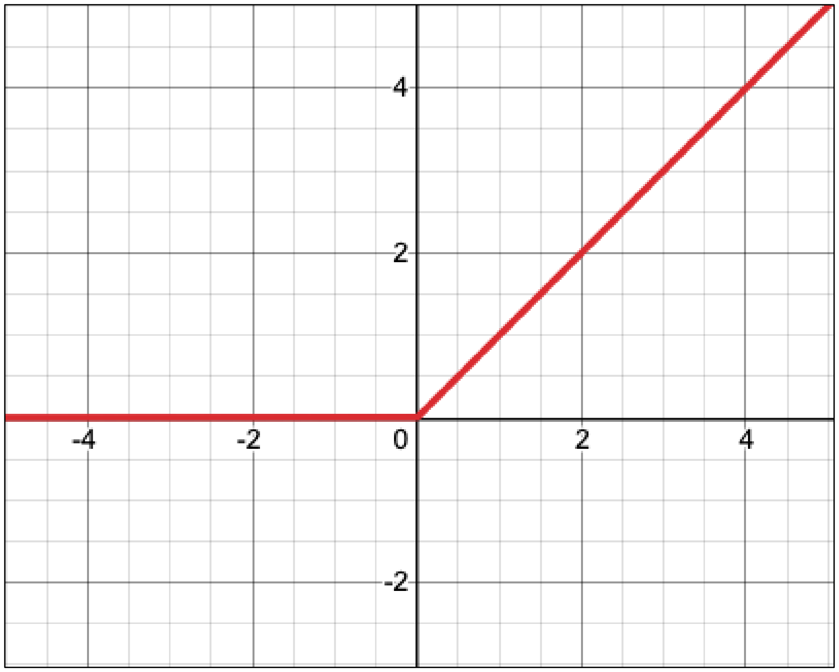

10
5
-5
0
5

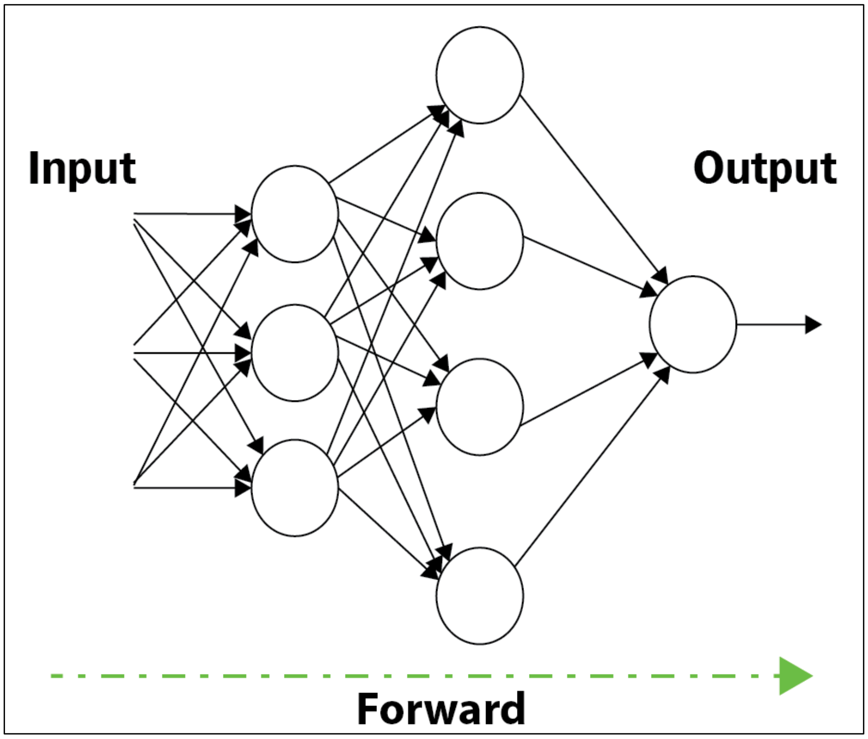

x_4
x_3
x_2
x_1
x_0

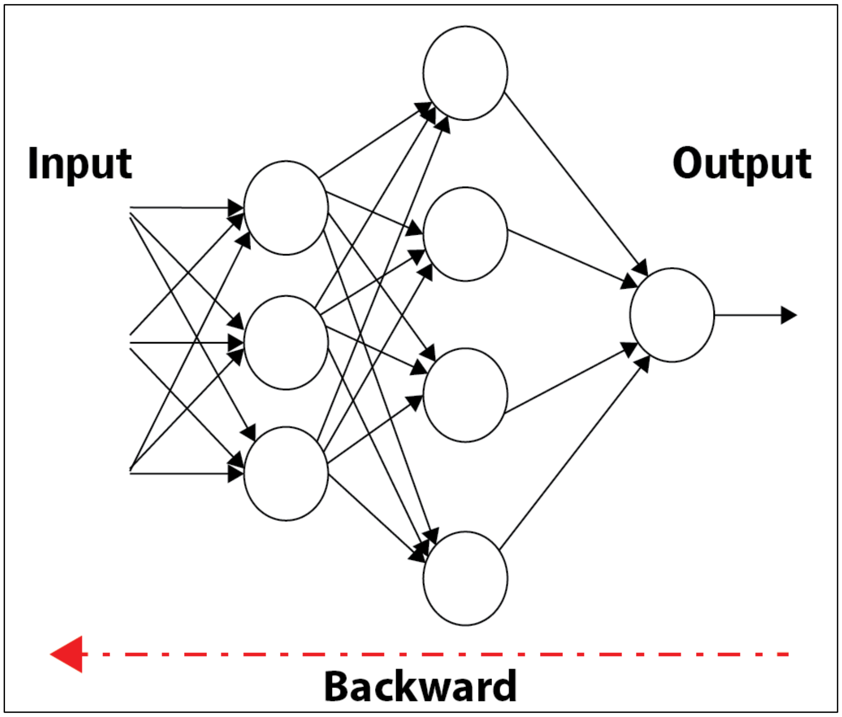

w_ij

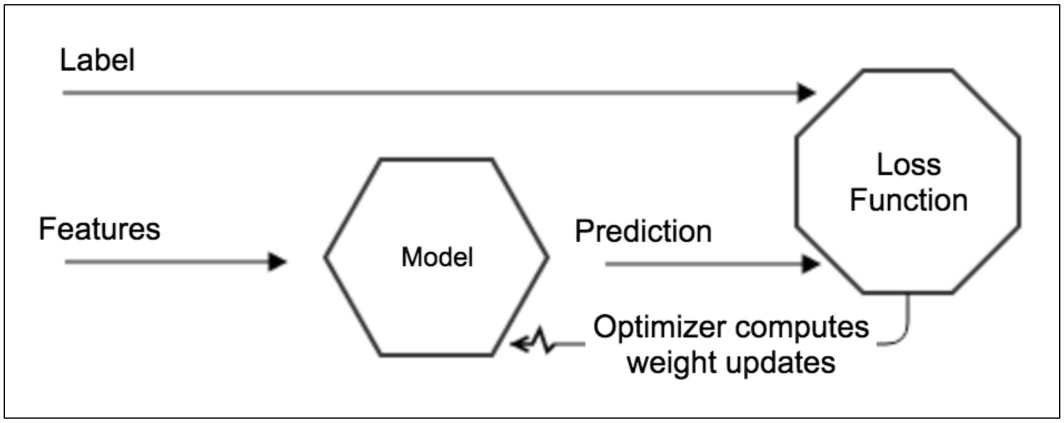

-5
0
5

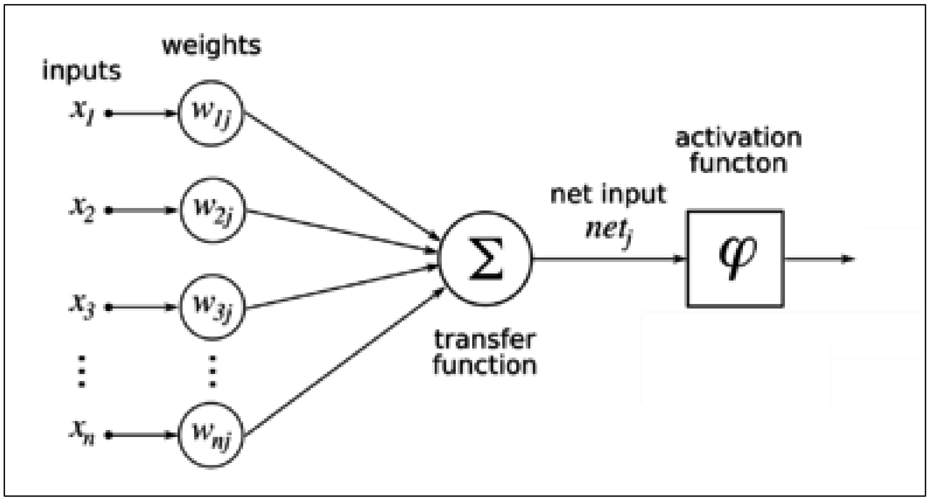

-5
5
0

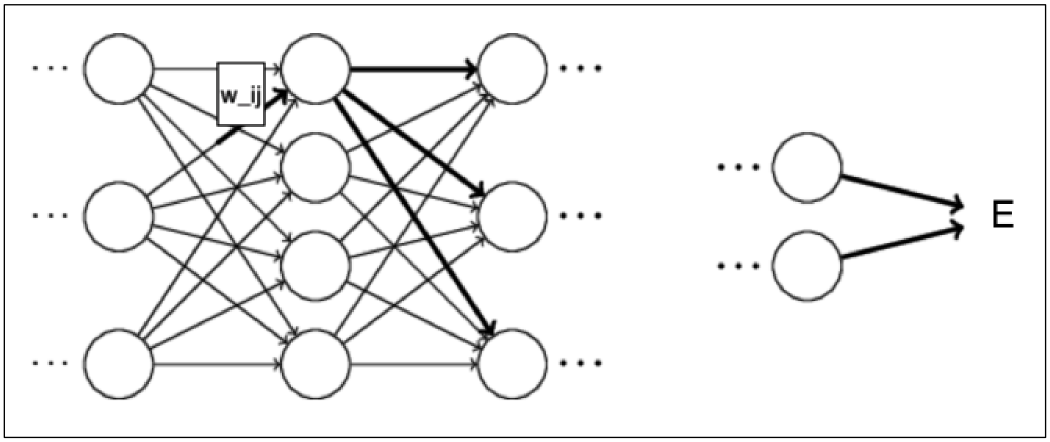

4
2
-2
-4
-2
0
2
4

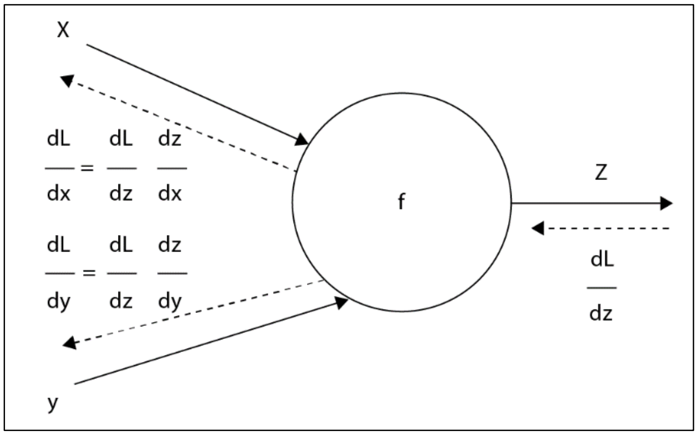

Input
Output
Forward

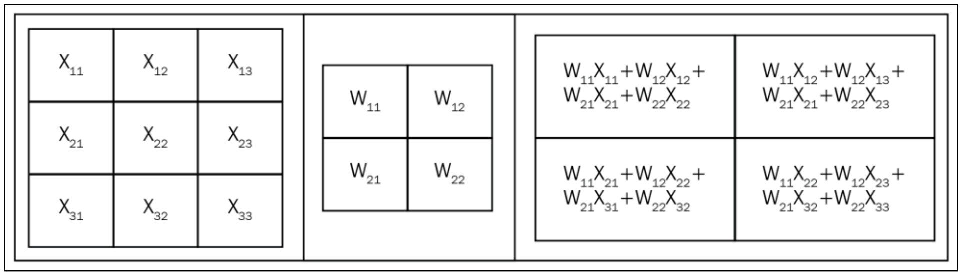

Input
Output
Backward


Label
Loss
Function
Features
Model
Prediction
Optimizer computes
weight updates


inputs
weights
x_1
x_2
x_3
x_n
w_{1j}
w_{2j}
w_{3j}
w_{nj}
Σ
transfer function
net input
net_j
activation functon
φ


w_ij
E

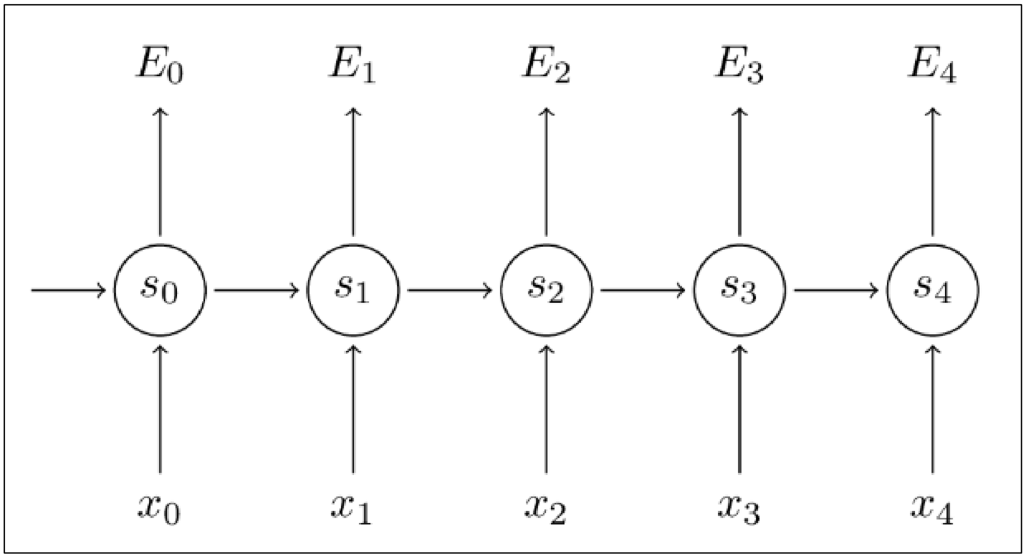

X
$\frac{dL}{dx} = \frac{dL}{dz} \frac{dz}{dx}$
$\frac{dL}{dy} = \frac{dL}{dz} \frac{dz}{dy}$
f
Z
$\frac{dL}{dz}$
y

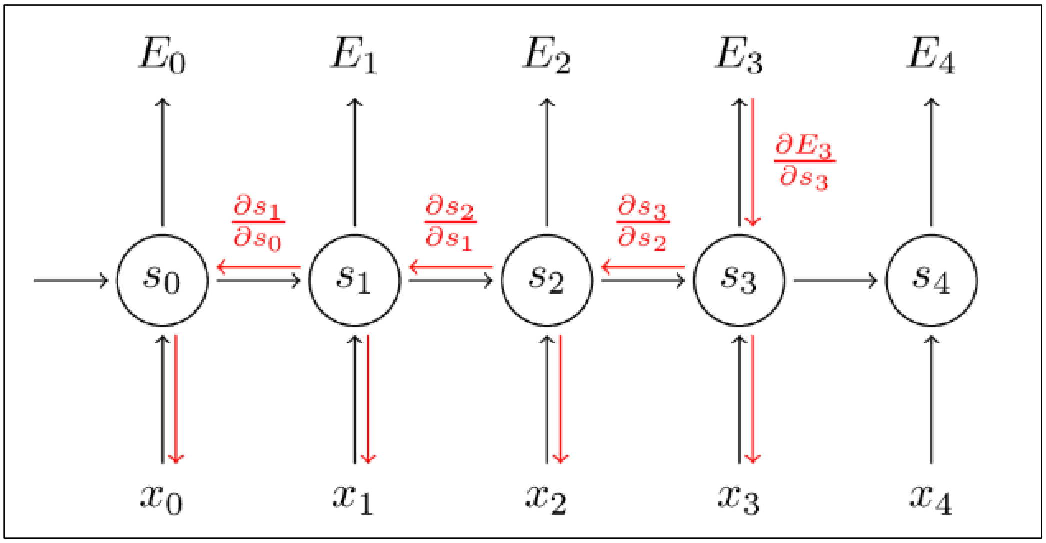


X_{11}	X_{12}	X_{13}
X_{21}	X_{22}	X_{23}
X_{31}	X_{32}	X_{33}

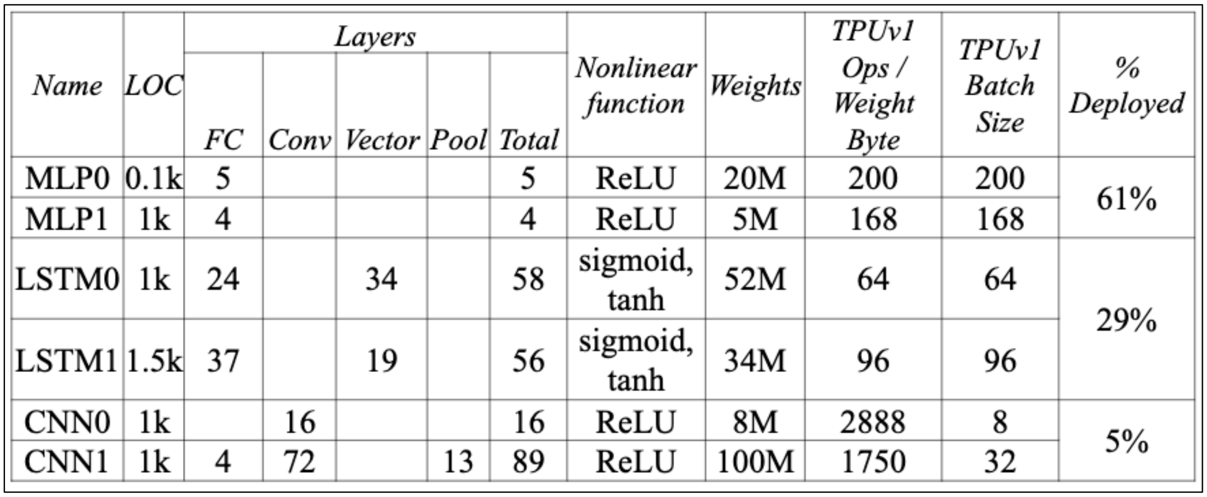
W_{11}	W_{12}
W_{21}	W_{22}

$W_{11}X_{11}+W_{12}X_{12}+$ $W_{21}X_{21}+W_{22}X_{22}$	$W_{11}X_{12}+W_{12}X_{13}+$ $W_{21}X_{21}+W_{22}X_{23}$
$W_{11}X_{21}+W_{12}X_{22}+$ $W_{21}X_{31}+W_{22}X_{32}$	$W_{11}X_{22}+W_{12}X_{23}+$ $W_{21}X_{32}+W_{22}X_{33}$

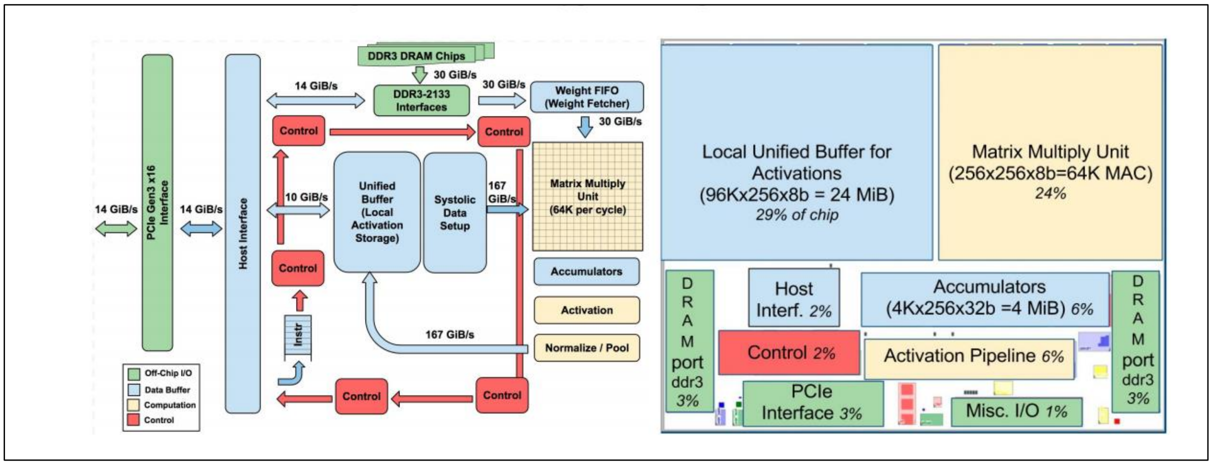


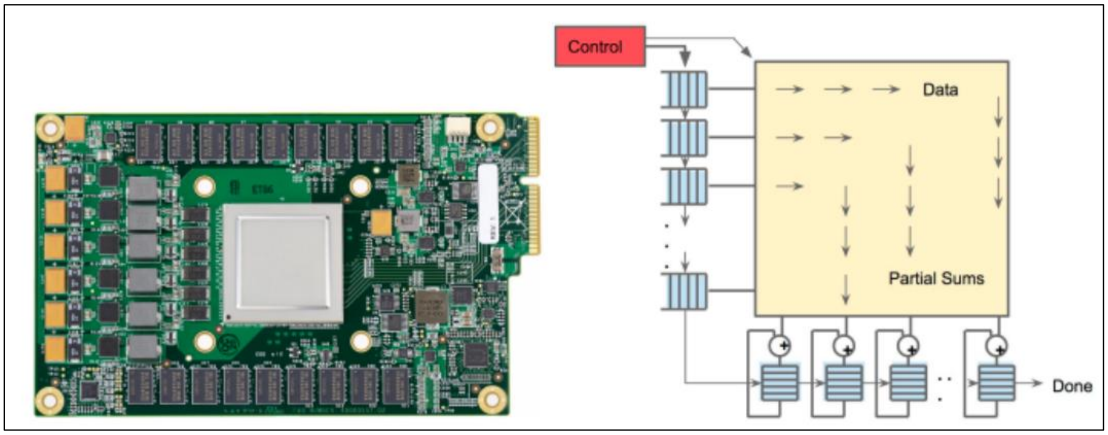

E_0
E_1
E_2
E_3
E_4
$\frac{\partial E_3}{\partial s_3}$
$\frac{\partial s_1}{\partial s_0}$
$\frac{\partial s_2}{\partial s_1}$
$\frac{\partial s_3}{\partial s_2}$
s_0
s_1
s_2
s_3
s_4
x_0
x_1
x_2
x_3
x_4

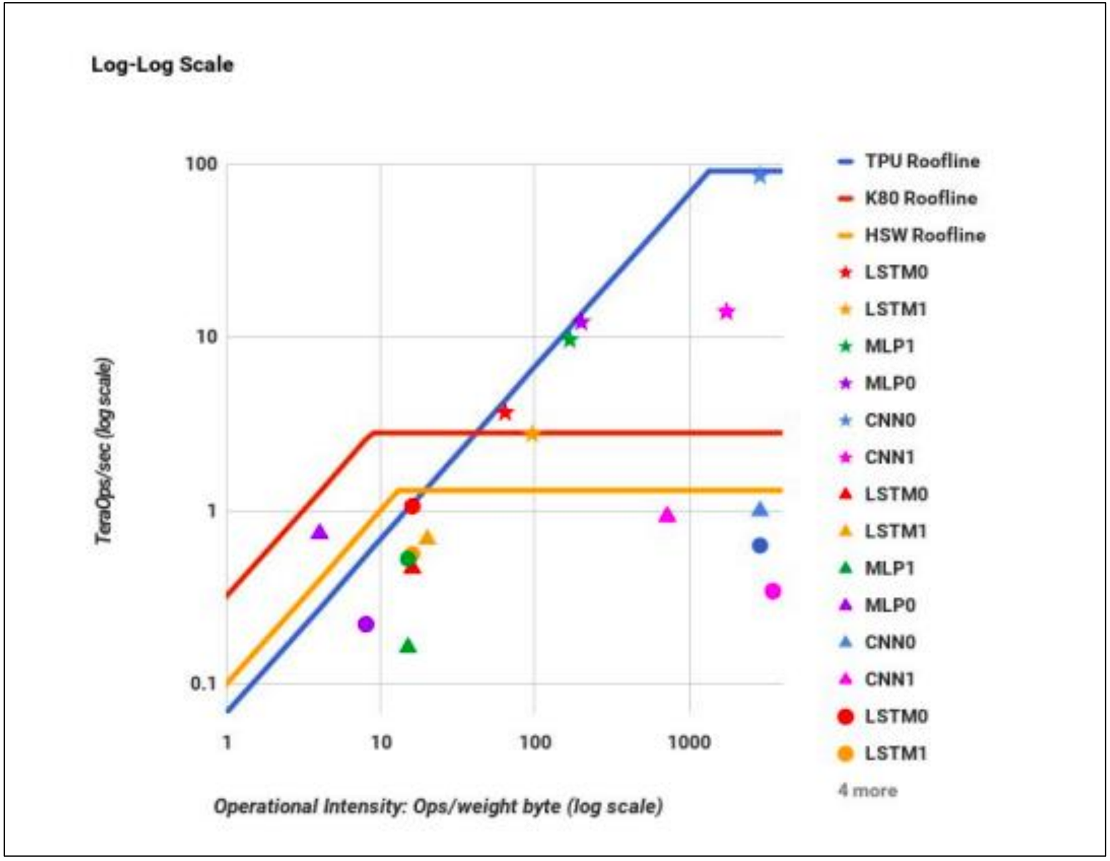
Chapter 16: Tensor Processing Unit

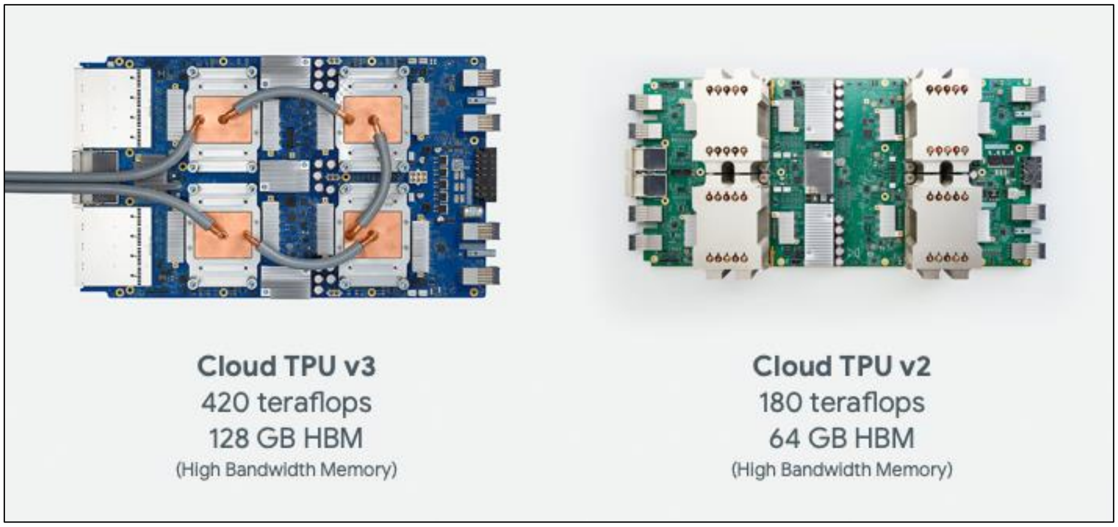


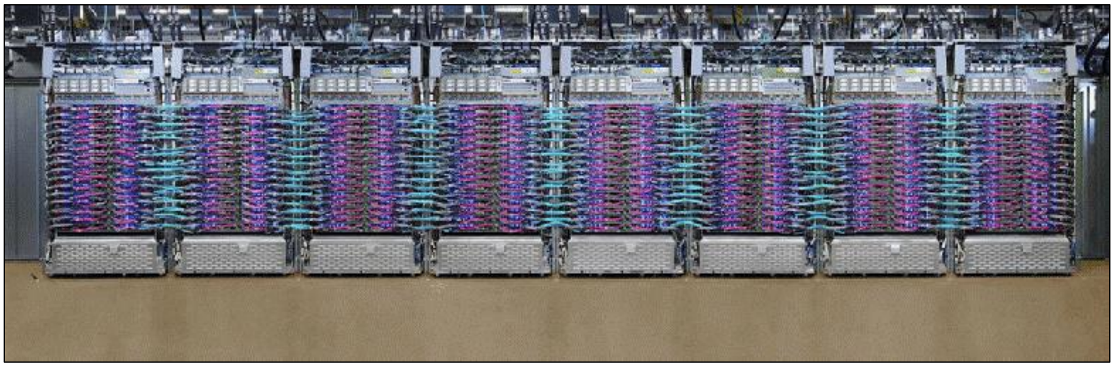
Name	*LOC*	*Layers*					*Nonlinear function*	*Weights*	*TPUv1 Ops / Weight Byte*	*TPUv1 Batch Size*	*% Deployed*
		FC	*Conv*	*Vector*	*Pool*	*Total*					
MLP0	0.1k	5				5	ReLU	20M	200	200	61%
MLP1	1k	4				4	ReLU	5M	168	168	
LSTM0	1k	24		34		58	sigmoid, tanh	52M	64	64	29%
LSTM1	1.5k	37		19		56	sigmoid, tanh	34M	96	96	
CNN0	1k		16			16	ReLU	8M	2888	8	5%
CNN1	1k	4	72		13	89	ReLU	100M	1750	32	

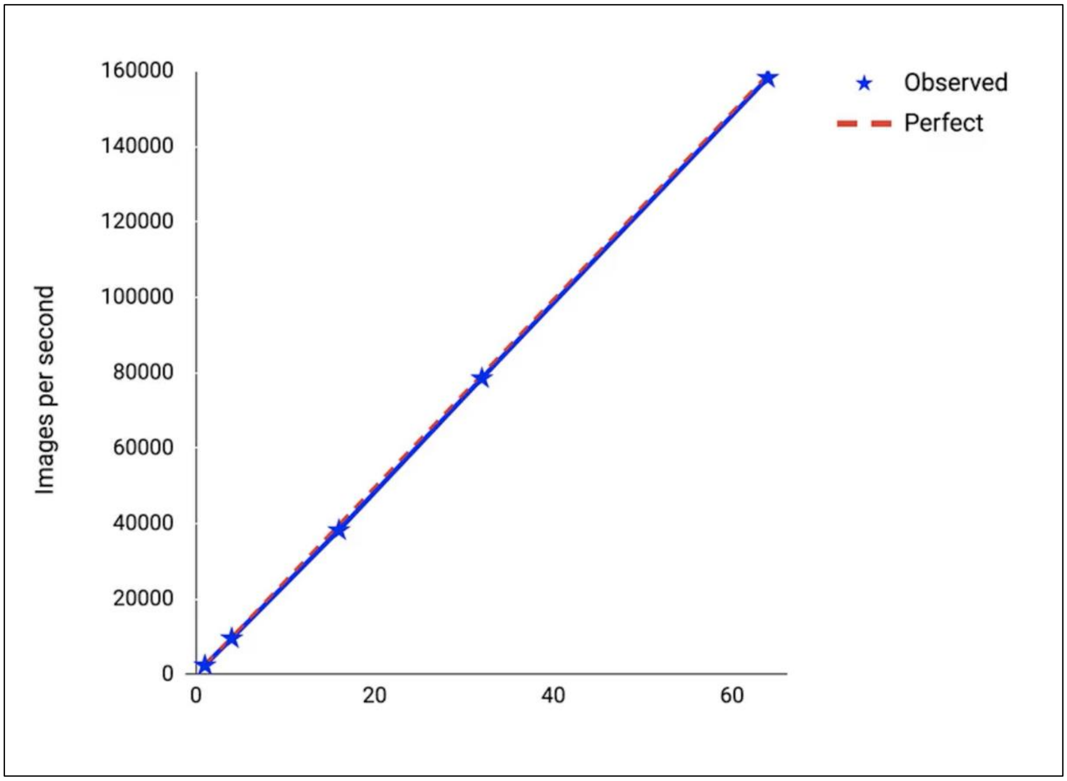


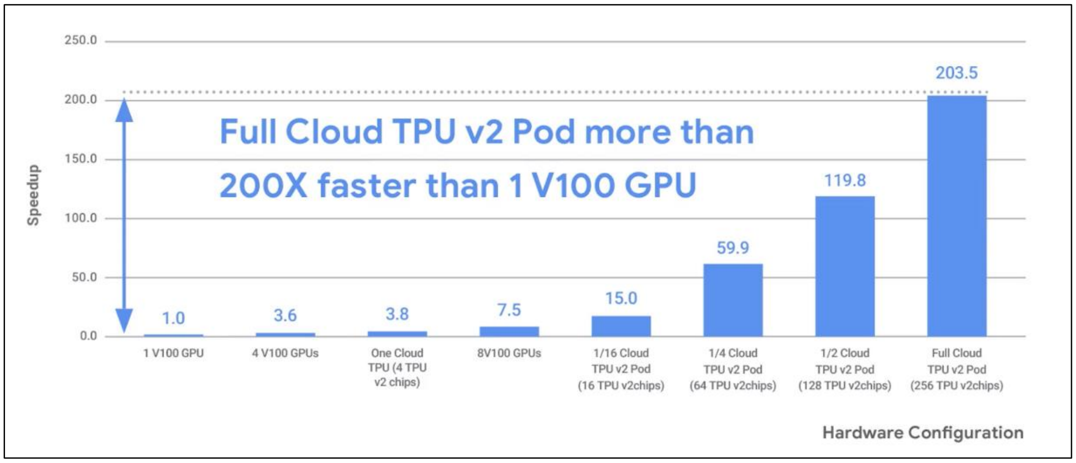

Control
Data
Partial Sums
Done

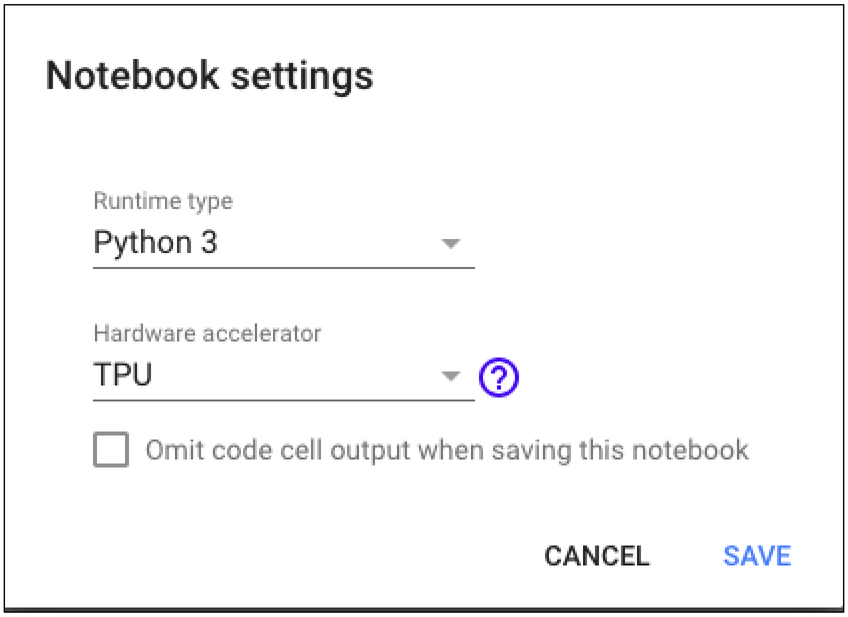

Log-Log Scale
100
10
1
0.1
1
10
100
1000
TeraOps/sec (log scale)
Operational Intensity: Ops/weight byte (log scale)
TPU Roofline
K80 Roofline
HSW Roofline
★ LSTM0
★ LSTM1
★ MLP1
★ MLP0
★ CNN0
★ CNN1
▲ LSTM0
▲ LSTM1
▲ MLP1
▲ MLP0
▲ CNN0
▲ CNN1
● LSTM0
● LSTM1
4 more


Cloud TPU v3
420 teraflops
128 GB HBM
(High Bandwidth Memory)
Cloud TPU v2
180 teraflops
64 GB HBM
(High Bandwidth Memory)






Full Cloud TPU v2 Pod more than 200X faster than 1 V100 GPU

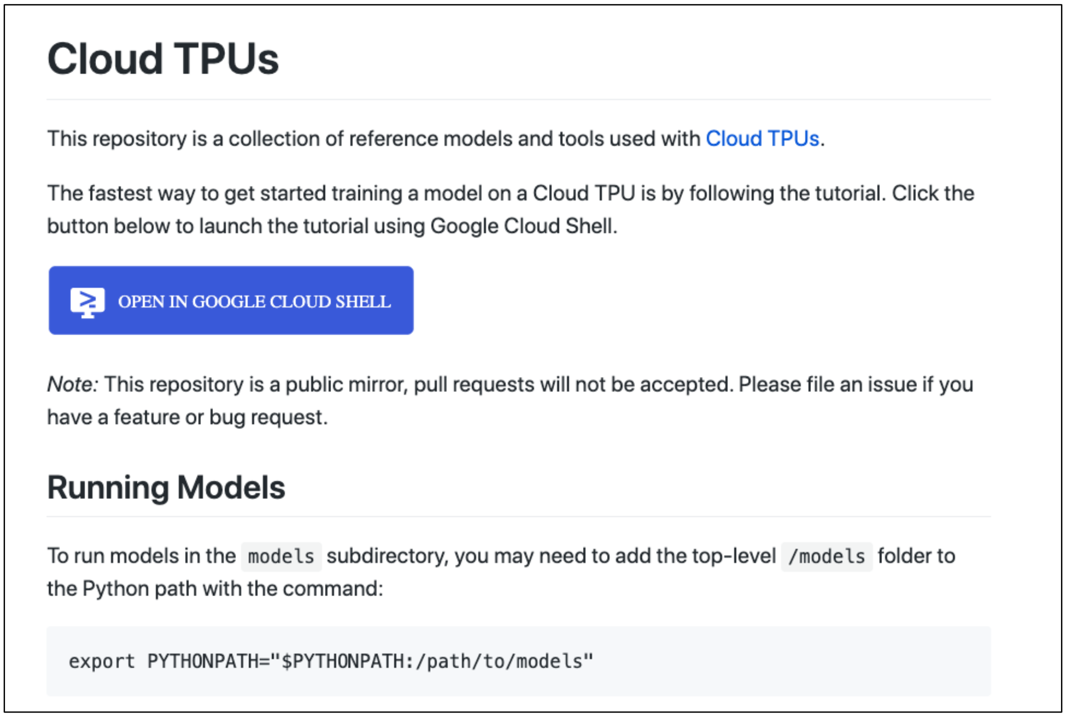


Notebook settings

Runtime type
Python 3

Hardware accelerator
TPU

□ Omit code cell output when saving this notebook



Cloud TPUs

This repository is a collection of reference models and tools used with Cloud TPUs.

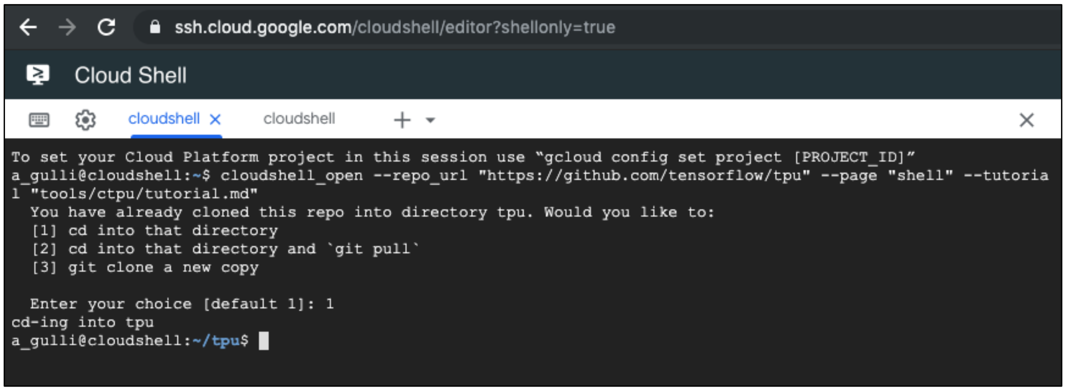
The fastest way to get started training a model on a Cloud TPU is by following the tutorial. Click the button below to launch the tutorial using Google Cloud Shell.

Note: This repository is a public mirror, pull requests will not be accepted. Please file an issue if you have a feature or bug request.

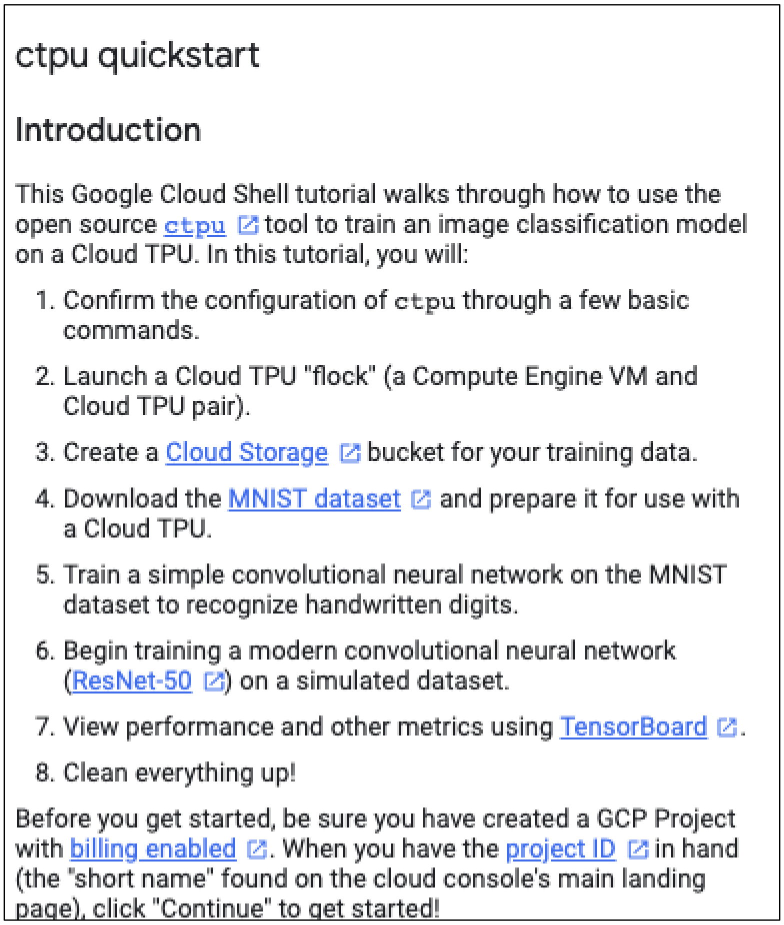
Running Models

To run models in the `models` subdirectory, you may need to add the top-level `/models` folder to the Python path with the command:

```
export PYTHONPATH="$PYTHONPATH:/path/to/models"
```



```
To set your Cloud Platform project in this session use "gcloud config set project [PROJECT_ID]"
a_gulli@cloudshell:~$ cloudshell_open --repo_url "https://github.com/tensorflow/tpu" --page "shell" --tutoria
l "tools/ctpu/tutorial.md"
  You have already cloned this repo into directory tpu. Would you like to:
  [1] cd into that directory
  [2] cd into that directory and `git pull`
  [3] git clone a new copy

  Enter your choice [default 1]: 1
cd-ing into tpu
a_gulli@cloudshell:~/tpu$
```



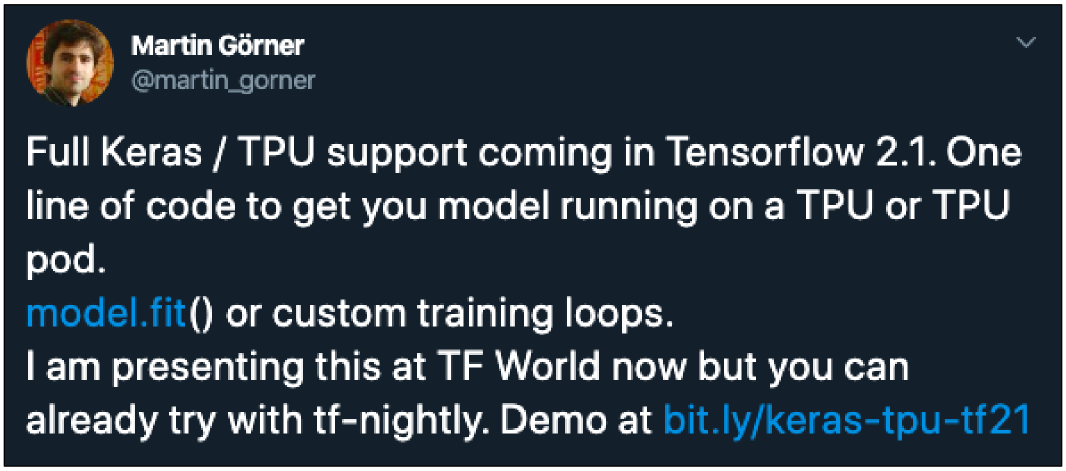
ctpu quickstart

Introduction

This Google Cloud Shell tutorial walks through how to use the open source `ctpu` tool to train an image classification model on a Cloud TPU. In this tutorial, you will:

1. Confirm the configuration of `ctpu` through a few basic commands.
2. Launch a Cloud TPU "flock" (a Compute Engine VM and Cloud TPU pair).
3. Create a Cloud Storage bucket for your training data.
4. Download the MNIST dataset and prepare it for use with a Cloud TPU.
5. Train a simple convolutional neural network on the MNIST dataset to recognize handwritten digits.
6. Begin training a modern convolutional neural network (ResNet-50) on a simulated dataset.
7. View performance and other metrics using TensorBoard.
8. Clean everything up!

Before you get started, be sure you have created a GCP Project with billing enabled. When you have the project ID in hand (the "short name" found on the cloud console's main landing page), click "Continue" to get started!



Martin Görner
@martin_gorner

Full Keras / TPU support coming in Tensorflow 2.1. One line of code to get you model running on a TPU or TPU pod.
model.fit() or custom training loops.
I am presenting this at TF World now but you can already try with tf-nightly. Demo at bit.ly/keras-tpu-tf21