```
!pip install PyDrive
```

```
!pip install keras
```

```
Requirement already satisfied: keras in /usr/local/lib/python3.7/dist-packages (2.7.0)
```

```
!pip install tensorflow
```

```
!pip install Tensorboard
```

```
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
import seaborn as sns
import numpy as np
from datetime import datetime as dt
import matplotlib.pyplot as plt
from sklearn.preprocessing import OneHotEncoder
from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import xgboost as xgb
from xgboost import XGBClassifier
import math
import tensorflow.keras as keras
from keras.models import Sequential
from keras.layers import Dense
import tensorflow.keras as keras
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
import geopy.distance
```

```
# 1. Authenticate and create the PyDrive client.
auth.authenticate_user()
gauth = GoogleAuth()
```

```
gauth.credentials = GoogleCredentials.get_application_default()
```

```
#2. Get the file
downloaded = drive.CreateFile({'id':"17TlCXfs4WKyo8nshLrTGoVTLOvP6CIyr"})   # replace the id
downloaded.GetContentFile('fraudTrain.csv')
```

```
#3. Read file as panda dataframe
import pandas as pd
data = pd.read_csv('fraudTrain.csv')
```

```
data.shape
```

```
data['trans_date_trans_time'] = pd.to_datetime(data['trans_date_trans_time'])
data.dtypes['trans_date_trans_time']

data['transaction_hour'] = data['trans_date_trans_time'].dt.hour

t = data.groupby('transaction_hour').count()
t = t['trans_num']
t = pd.DataFrame(t)
t = t.rename(columns={'trans_num':'Actual_transaction'})

fraud_t = data.loc[data['is_fraud']==1]
tf = fraud_t.groupby('transaction_hour').count()
tf = tf['trans_num']

time = pd.concat([t, tf], axis=1)
time[['Actual_transaction','trans_num']]

time = time.rename(columns={'trans_num':'Fradulent_transaction'})

time['Fradulent_transaction'] = time['Fradulent_transaction'].fillna(0)

time['fraud_rate%_by_hour'] = (time['Fradulent_transaction'] / time['Actual_transaction']) *

time = time.sort_values(['fraud_rate%_by_hour'], ascending=False)
fr_time = pd.DataFrame(time['fraud_rate%_by_hour'])

act_time = data['transaction_hour']
act_time = pd.DataFrame(act_time)

new_time = pd.merge(act_time, fr_time, how='left', on='transaction_hour')
new_time
```

```
data['transaction_hour'] = new_time['fraud_rate%_by_hour']
```

```
t_categ_anal = data.groupby('category')[['trans_num']].count().reset_index()
t_categ_anal.rename({'trans_num':'total_count_of_trasactions'}, axis=1)
```

```
f_categ_anal = data[data['is_fraud']==1].groupby('category')[['trans_num']].count().reset_ind
f_categ_anal.rename({'trans_num':'count_of_fraud_trasactions'}, axis=1)

categ_anal = pd.merge(t_categ_anal, f_categ_anal, how='left', on='category')
categ_anal['fraud_perc'] = (categ_anal['trans_num_y'] / categ_anal['trans_num_x']) * 100
categ_anal = categ_anal.sort_values(['fraud_perc'], ascending=False)

t_job_anal = data.groupby('job')[['trans_num']].count().reset_index()
t_job_anal = t_job_anal.sort_values(['trans_num'], ascending=False)

f_job_anal = data[data['is_fraud']==1].groupby('job')[['trans_num']].count().reset_index()
f_job_anal = f_job_anal.sort_values(['trans_num'], ascending=False)

job_anal = pd.merge(t_job_anal, f_job_anal, how='inner', on='job')
job_anal['fraud_perc'] = (job_anal['trans_num_y'] / job_anal['trans_num_x']) * 100
job_anal = job_anal.sort_values(['fraud_perc'], ascending=False)
high_fraud = job_anal[job_anal['fraud_perc'] > 90]
#high_fraud = pd.DataFrame(high_fraud['job'])
high_fraud_job = high_fraud['job'].tolist()

new_job = []

for job in data['job']:
  for fraud_job in high_fraud_job:
    if fraud_job == job:
      new_job.append('high risk')
      break
  else:
    new_job.append('low risk')

data['job'] = new_job


enc = OneHotEncoder()
enc_df = pd.DataFrame(enc.fit_transform(data[['job']]).toarray())
enc_df
data = data.join(enc_df)


data.rename({0:'low risk job', 1:'high risk job'}, axis=1, inplace=True)


z = data.groupby('zip').count()
z = z['city']
z = pd.DataFrame(z)
z = z.rename(columns={'city':'Actual_transaction'})

fraud = data.loc[data['is_fraud']==1]
zf = fraud.groupby('zip').count()
zf = zf['city']
```

```
zip = pd.concat([z, zf], axis=1)
zip[['Actual_transaction','city']]

zip = zip.rename(columns={'city':'Fradulent_transaction'})

zip['Fradulent_transaction'] = zip['Fradulent_transaction'].fillna(0)

zip['fraud_rate%'] = (zip['Fradulent_transaction'] / zip['Actual_transaction']) * 100

zip = zip.sort_values(['fraud_rate%'], ascending=False)
fr_zip = pd.DataFrame(zip['fraud_rate%'])

# Replacing Zip codes with fraud rates
act_zip = data['zip']
act_zip = pd.DataFrame(act_zip)

new_zip = pd.merge(act_zip, fr_zip, how='left', on='zip')

data['zip'] = new_zip['fraud_rate%']
data['zip']


data['amt'] = np.log(data['amt'])
data['amt'].skew()


enc = OneHotEncoder()
enc_df = pd.DataFrame(enc.fit_transform(data[['gender']]).toarray())
enc_df
data = data.join(enc_df)


data.rename({0:'female', 1:'male'}, axis=1, inplace=True)


data['category'].replace({'misc_net':1, 'grocery_pos':2, 'entertainment':3, 'gas_transport':4


enc = OneHotEncoder()
enc_df = pd.DataFrame(enc.fit_transform(data[['category']]).toarray())
enc_df
data = data.join(enc_df)


data.rename({0:'misc_net', 1:'grocery_pos', 2:'entertainment', 3:'gas_transport', 4:'misc_pos


data['day_of_week'] = data['trans_date_trans_time'].dt.day_name()
data['day_of_week']


label_encoder = preprocessing.LabelEncoder()
data['day_of_week'] = label_encoder.fit_transform(data['day_of_week'])
```

```
data['day_of_week'].unique()
```

```
array([5, 6, 4, 0, 2, 3, 1])
```

```
data.columns
```

```
Index(['Unnamed: 0', 'trans_date_trans_time', 'cc_num', 'merchant', 'category',
       'amt', 'first', 'last', 'gender', 'street', 'city', 'state', 'zip',
       'lat', 'long', 'city_pop', 'job', 'dob', 'trans_num', 'unix_time',
       'merch_lat', 'merch_long', 'is_fraud', 'transaction_hour',
       'low risk job', 'high risk job', 'female', 'male', 'misc_net',
       'grocery_pos', 'entertainment', 'gas_transport', 'misc_pos',
       'grocery_net', 'shopping_net', 'shopping_pos', 'food_dining',
       'personal_care', 'health_fitness', 'travel', 'kids_pets', 'home',
       'day_of_week'],
      dtype='object')
```

```
Y = data['is_fraud']
```

```
data.drop(columns=['Unnamed: 0','is_fraud','trans_date_trans_time','dob','trans_num','unix_ti
```

```
X = data.copy()
```

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.30)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(907672, 21) (389003, 21) (907672,) (389003,)
```

```
sc = StandardScaler()
sc.fit(X_train)
```

```
X_train_normalized = sc.transform(X_train)
X_test_normalized = sc.transform(X_test)
```

```
# convert to Pandas DF
X_train_normalized = pd.DataFrame(X_train_normalized, columns=X_train.columns)
X_test_normalized = pd.DataFrame(X_test_normalized, columns=X_test.columns)
```

```
X_train_normalized.describe().apply(lambda s: s.apply('{0:.2f}'.format))
```

Feature Selection

```
xgb_instance = xgb.XGBClassifier()
```

```
model_for_feature_selection = xgb_instance.fit(X_train_normalized, y_train)
```

```
feature_importance = {'Feature':X_train_normalized.columns,'Importance':model_for_feature_sel
feature_importance = pd.DataFrame(feature_importance)
feature_importance.sort_values("Importance", inplace=True,ascending=False)
feature_importance
```

	Feature	Importance
10	misc_pos	0.305904
8	entertainment	0.211822
0	amt	0.084691
3	transaction_hour	0.079823
6	misc_net	0.060125
12	shopping_net	0.047648
17	travel	0.035314
15	personal_care	0.035209
1	zip	0.034306
9	gas_transport	0.029079
7	grocery_pos	0.023501
19	home	0.022225
18	kids_pets	0.019053
16	health_fitness	0.004523
13	shopping_pos	0.003835
2	city_pop	0.002943
5	male	0.000000
11	grocery_net	0.000000
4	high risk job	0.000000
14	food_dining	0.000000
20	day_of_week	0.000000

```
final_features = feature_importance["Feature"][feature_importance.Importance > 0.01]
```

```
X_train_normalized = X_train_normalized[final_features]
X_test_normalized = X_test_normalized[final_features]
```

```
X_train_normalized.head(5)

X_test_normalized.head(5)
```

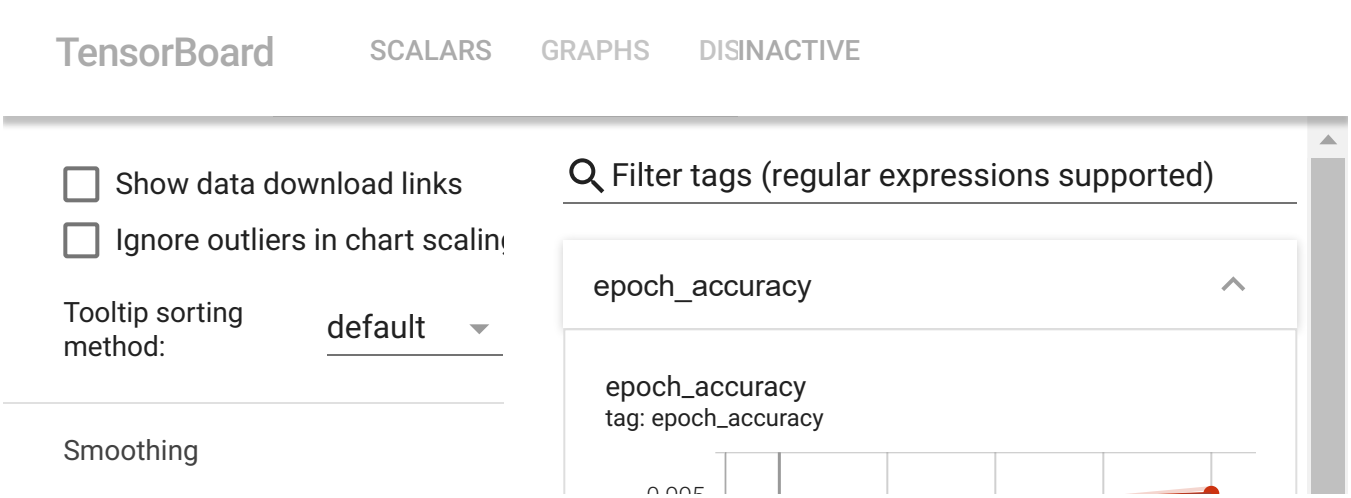
MODEL

```
classifier = Sequential()

# add the first hidden layer
classifier.add(Dense(units=5,kernel_initializer='glorot_uniform',
                     activation = 'relu'))

# add the second hidden layer
classifier.add(Dense(units=5,kernel_initializer='glorot_uniform',
                     activation = 'relu'))

# add the output layer
classifier.add(Dense(units=1,kernel_initializer='glorot_uniform',
                     activation = 'sigmoid'))

# add additional parameters
classifier.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])

# train the model
classifier.fit(X_train_normalized,y_train,batch_size=1000,epochs=20)


def build_classifier():
    classifier = Sequential()
    classifier.add(Dense(units=5,kernel_initializer='glorot_uniform', activation = 'relu'))
    classifier.add(Dense(units=5,kernel_initializer='glorot_uniform', activation = 'relu'))
    classifier.add(Dense(units=1,kernel_initializer='glorot_uniform', activation = 'sigmoid'))
    classifier.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
    #classifier.fit(X_train_normalized,y_train,batch_size=1000,epochs=20)
    return classifier


model=KerasClassifier(build_fn=build_classifier)


import tensorflow as tf
import datetime


rm -rf ./logs/


log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)
```

```
model.fit(x=X_train_normalized,
          y=y_train,
          batch_size=1000,
          epochs=5,
          validation_data=(X_test_normalized, y_test),
          callbacks=[tensorboard_callback])
```

```
Epoch 1/5
908/908 [==============================] - 3s 3ms/step - loss: 0.2459 - accuracy: 0.9160
Epoch 2/5
908/908 [==============================] - 2s 2ms/step - loss: 0.0256 - accuracy: 0.9948
Epoch 3/5
908/908 [==============================] - 2s 2ms/step - loss: 0.0219 - accuracy: 0.9948
Epoch 4/5
908/908 [==============================] - 2s 2ms/step - loss: 0.0206 - accuracy: 0.9948
Epoch 5/5
908/908 [==============================] - 2s 2ms/step - loss: 0.0189 - accuracy: 0.9952
<keras.callbacks.History at 0x7f3b00a97f10>
```

```
%tensorboard --logdir logs/fit
```




```
params={'batch_size':[100, 20],
        'nb_epoch':[20, 10],
        'unit':[5,6],

        }
gs=GridSearchCV(estimator=model, param_grid=params, cv=10)
# now fit the dataset to the GridSearchCV object.
gs = gs.fit(X_train, y_train)
```

```
Epoch 1/20
908/908 [==============================] - 3s 2ms/step - loss: 0.1100 - accuracy: 0.9
Epoch 2/20
908/908 [==============================] - 2s 2ms/step - loss: 0.0274 - accuracy: 0.9
Epoch 3/20
908/908 [==============================] - 2s 2ms/step - loss: 0.0256 - accuracy: 0.9
Epoch 4/20
908/908 [==============================] - 2s 2ms/step - loss: 0.0237 - accuracy: 0.9
Epoch 5/20
908/908 [==============================] - 2s 2ms/step - loss: 0.0186 - accuracy: 0.9
Epoch 6/20
908/908 [==============================] - 2s 2ms/step - loss: 0.0171 - accuracy: 0.9
Epoch 7/20
908/908 [==============================] - 2s 3ms/step - loss: 0.0158 - accuracy: 0.9
Epoch 8/20
908/908 [==============================] - 2s 2ms/step - loss: 0.0147 - accuracy: 0.9
Epoch 9/20
908/908 [==============================] - 2s 2ms/step - loss: 0.0136 - accuracy: 0.9
Epoch 10/20
908/908 [==============================] - 2s 2ms/step - loss: 0.0127 - accuracy: 0.9
Epoch 11/20
908/908 [==============================] - 2s 2ms/step - loss: 0.0118 - accuracy: 0.9
Epoch 12/20
908/908 [==============================] - 2s 2ms/step - loss: 0.0111 - accuracy: 0.9
Epoch 13/20
908/908 [==============================] - 2s 2ms/step - loss: 0.0104 - accuracy: 0.9
Epoch 14/20
908/908 [==============================] - 2s 3ms/step - loss: 0.0098 - accuracy: 0.9
Epoch 15/20
908/908 [==============================] - 2s 2ms/step - loss: 0.0094 - accuracy: 0.9
Epoch 16/20
```

```
908/908 [==============================] - 2s 2ms/step - loss: 0.0091 - accuracy: 0.9
Epoch 17/20
908/908 [==============================] - 2s 2ms/step - loss: 0.0089 - accuracy: 0.9
Epoch 18/20
908/908 [==============================] - 2s 2ms/step - loss: 0.0087 - accuracy: 0.9
Epoch 19/20
908/908 [==============================] - 2s 2ms/step - loss: 0.0085 - accuracy: 0.9
Epoch 20/20
908/908 [==============================] - 2s 2ms/step - loss: 0.0084 - accuracy: 0.9
Epoch 1/20
908/908 [==============================] - 3s 2ms/step - loss: 0.1123 - accuracy: 0.9
Epoch 2/20
908/908 [==============================] - 2s 2ms/step - loss: 0.0231 - accuracy: 0.9
Epoch 3/20
908/908 [==============================] - 2s 2ms/step - loss: 0.0213 - accuracy: 0.9
Epoch 4/20
908/908 [==============================] - 2s 2ms/step - loss: 0.0201 - accuracy: 0.9
Epoch 5/20
908/908 [==============================] - 2s 2ms/step - loss: 0.0189 - accuracy: 0.9
Epoch 6/20
908/908 [==============================] - 2s 2ms/step - loss: 0.0179 - accuracy: 0.9
Epoch 7/20
908/908 [==============================] - 2s 2ms/step - loss: 0.0172 - accuracy: 0.9
Epoch 8/20
908/908 [==============================] - 2s 2ms/step - loss: 0.0166 - accuracy: 0.9
Epoch 9/20
```

Hyperparameter Tuning

```
%load_ext tensorboard
```

```
rm -rf ./logs/
```

```
import tensorflow as tf
from tensorboard.plugins.hparams import api as hp
import datetime
```

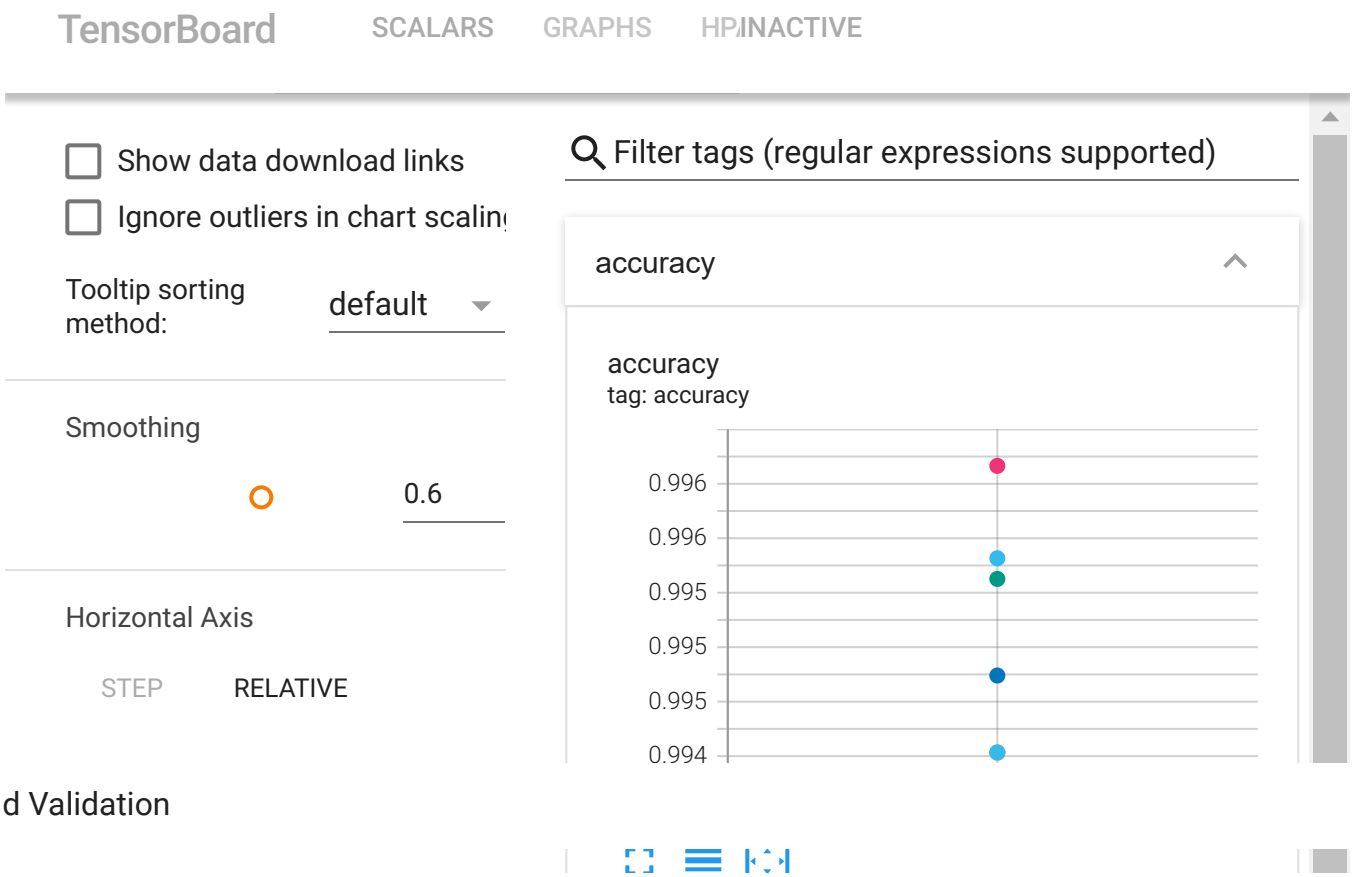
```
HP_NUM_UNITS = hp.HParam('num_units', hp.Discrete([2, 34]))
HP_DROPOUT = hp.HParam('dropout', hp.RealInterval(0.1, 0.2))
HP_OPTIMIZER = hp.HParam('optimizer', hp.Discrete(['adam', 'sgd']))

METRIC_ACCURACY = 'accuracy'
hparams=[HP_NUM_UNITS, HP_DROPOUT, HP_OPTIMIZER]

with tf.summary.create_file_writer('logs/hparam_tuning').as_default():
  hp.hparams_config(
    hparams=[HP_NUM_UNITS, HP_DROPOUT, HP_OPTIMIZER],
```

```
      metrics=[hp.Metric(METRIC_ACCURACY, display_name='Accuracy')],
    )
  def train_test_model(hparams):
    model = tf.keras.models.Sequential([
      tf.keras.layers.Flatten(),
      tf.keras.layers.Dense(hparams[HP_NUM_UNITS], activation=tf.nn.relu),
      tf.keras.layers.Dropout(hparams[HP_DROPOUT]),
      tf.keras.layers.Dense(10, activation=tf.nn.softmax),
    ])
    model.compile(
        optimizer=hparams[HP_OPTIMIZER],
        loss='binary_crossentropy',
        metrics=['accuracy'],
    )

    log_dir = "logs/hparam_tuning" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
    tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir), hp.KerasCallback(lo

    model.fit(X_train_normalized, y_train, epochs=1, callbacks=[tensorboard_callback])
    # Run with 1 epoch to speed things up for demo purposes
    _,accuracy = model.evaluate(X_test_normalized, y_test)
    return accuracy


  def run(run_dir, hparams):
    with tf.summary.create_file_writer(run_dir).as_default():
      hp.hparams(hparams)  # record the values used in this trial
      accuracy = train_test_model(hparams)
      tf.summary.scalar(METRIC_ACCURACY, accuracy, step=1)


  session_num = 0

  for num_units in HP_NUM_UNITS.domain.values:
    for dropout_rate in (HP_DROPOUT.domain.min_value, HP_DROPOUT.domain.max_value):
      for optimizer in HP_OPTIMIZER.domain.values:
        hparams = {
            HP_NUM_UNITS: num_units,
            HP_DROPOUT: dropout_rate,
            HP_OPTIMIZER: optimizer,
        }
        run_name = "run-%d" % session_num
        print('--- Starting trial: %s' % run_name)
        print({h.name: hparams[h] for h in hparams})
        run('logs/hparam_tuning/' + run_name, hparams)
        session_num += 1

       --- Starting trial: run-0
       {'num_units': 2, 'dropout': 0.1, 'optimizer': 'adam'}
       28365/28365 [==============================] - 40s 1ms/step - loss: 0.0617 - accuracy: 0
       12157/12157 [==============================] - 14s 1ms/step - loss: 0.0252 - accuracy: 0
       --- Starting trial: run-1
```

```
{'num_units': 2, 'dropout': 0.1, 'optimizer': 'sgd'}
28365/28365 [==============================] - 38s 1ms/step - loss: 0.0522 - accuracy: 0
12157/12157 [==============================] - 15s 1ms/step - loss: 0.0276 - accuracy: 0
--- Starting trial: run-2
{'num_units': 2, 'dropout': 0.2, 'optimizer': 'adam'}
28365/28365 [==============================] - 41s 1ms/step - loss: 0.0895 - accuracy: 0
12157/12157 [==============================] - 15s 1ms/step - loss: 0.0301 - accuracy: 0
--- Starting trial: run-3
{'num_units': 2, 'dropout': 0.2, 'optimizer': 'sgd'}
28365/28365 [==============================] - 38s 1ms/step - loss: 0.0578 - accuracy: 0
12157/12157 [==============================] - 15s 1ms/step - loss: 0.0280 - accuracy: 0
--- Starting trial: run-4
{'num_units': 34, 'dropout': 0.1, 'optimizer': 'adam'}
28365/28365 [==============================] - 48s 2ms/step - loss: 0.0218 - accuracy: 0
12157/12157 [==============================] - 15s 1ms/step - loss: 0.0105 - accuracy: 0
--- Starting trial: run-5
{'num_units': 34, 'dropout': 0.1, 'optimizer': 'sgd'}
28365/28365 [==============================] - 39s 1ms/step - loss: 0.0322 - accuracy: 0
12157/12157 [==============================] - 14s 1ms/step - loss: 0.0182 - accuracy: 0
--- Starting trial: run-6
{'num_units': 34, 'dropout': 0.2, 'optimizer': 'adam'}
28365/28365 [==============================] - 47s 2ms/step - loss: 0.0250 - accuracy: 0
12157/12157 [==============================] - 15s 1ms/step - loss: 0.0118 - accuracy: 0
--- Starting trial: run-7
{'num_units': 34, 'dropout': 0.2, 'optimizer': 'sgd'}
28365/28365 [==============================] - 41s 1ms/step - loss: 0.0339 - accuracy: 0
12157/12157 [==============================] - 15s 1ms/step - loss: 0.0183 - accuracy: 0
```

```
%tensorboard --logdir logs/hparam_tuning
```

```
Reusing TensorBoard on port 6006 (pid 395), started 0:47:15 ago. (Use '!kill 395' to kil
```



K Fold Validation

```
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
from tensorflow.keras.losses import binary_crossentropy
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import KFold
import numpy as np

# Model configuration
batch_size = 50
img_width, img_height, img_num_channels = 32, 32, 3
loss_function = binary_crossentropy
no_classes = 100
no_epochs = 10
optimizer = Adam()
verbosity = 1
num_folds = 5

# Parse numbers as floats
input_train = X_train_normalized.astype('float32')
input_test = X_test_normalized.astype('float32')

# Normalize data
input_train = input_train / 255
input_test = input_test / 255
```

```
# Define per-fold score containers
acc_per_fold = []
loss_per_fold = []

# Merge inputs and targets
inputs = np.concatenate((input_train, input_test), axis=0)
targets = np.concatenate((y_train, y_test), axis=0)

# Define the K-fold Cross Validator
kfold = KFold(n_splits=num_folds, shuffle=True)

# K-fold Cross Validation model evaluation
fold_no = 1
for train, test in kfold.split(inputs, targets):

  # Define the model architecture

  model = Sequential()
  model.add(Dense(units=5,kernel_initializer='glorot_uniform', activation = 'relu'))
  model.add(Dense(units=5,kernel_initializer='glorot_uniform', activation = 'relu'))
  model.add(Dense(units=1,kernel_initializer='glorot_uniform', activation = 'sigmoid'))

  # Compile the model
  model.compile(loss=loss_function,
                optimizer=optimizer,
                metrics=['accuracy'])


  # Generate a print
  print('------------------------------------------------------------------------')
  print(f'Training for fold {fold_no} ...')

  # Fit data to model
  history = model.fit(inputs[train], targets[train],
              batch_size=batch_size,
              epochs=no_epochs,
              verbose=verbosity)

  # Generate generalization metrics
  scores = model.evaluate(inputs[test], targets[test], verbose=0)
  print(f'Score for fold {fold_no}: {model.metrics_names[0]} of {scores[0]}; {model.metrics_n
  acc_per_fold.append(scores[1] * 100)
  loss_per_fold.append(scores[0])

  # Increase fold number
  fold_no = fold_no + 1

# == Provide average scores ==
print('------------------------------------------------------------------------')
print('Score per fold')
```

```
for i in range(0, len(acc_per_fold)):
  print('------------------------------------------------------------------------')
  print(f'> Fold {i+1} - Loss: {loss_per_fold[i]} - Accuracy: {acc_per_fold[i]}%')
print('------------------------------------------------------------------------')
print('Average scores for all folds:')
print(f'> Accuracy: {np.mean(acc_per_fold)} (+- {np.std(acc_per_fold)})')
print(f'> Loss: {np.mean(loss_per_fold)}')
print('------------------------------------------------------------------------')
```

```
------------------------------------------------------------------------
Training for fold 1 ...
Epoch 1/10
20747/20747 [==============================] - 32s 1ms/step - loss: 0.0405 - accuracy
Epoch 2/10
20747/20747 [==============================] - 31s 1ms/step - loss: 0.0221 - accuracy
Epoch 3/10
20747/20747 [==============================] - 31s 1ms/step - loss: 0.0202 - accuracy
Epoch 4/10
20747/20747 [==============================] - 31s 1ms/step - loss: 0.0197 - accuracy
Epoch 5/10
20747/20747 [==============================] - 31s 1ms/step - loss: 0.0194 - accuracy
Epoch 6/10
20747/20747 [==============================] - 31s 1ms/step - loss: 0.0194 - accuracy
Epoch 7/10
20747/20747 [==============================] - 30s 1ms/step - loss: 0.0194 - accuracy
Epoch 8/10
20747/20747 [==============================] - 31s 1ms/step - loss: 0.0193 - accuracy
Epoch 9/10
20747/20747 [==============================] - 30s 1ms/step - loss: 0.0193 - accuracy
Epoch 10/10
20747/20747 [==============================] - 30s 1ms/step - loss: 0.0193 - accuracy
Score for fold 1: loss of 0.018657712265849113; accuracy of 99.57314133644104%
------------------------------------------------------------------------
Training for fold 2 ...
Epoch 1/10
20747/20747 [==============================] - 31s 1ms/step - loss: 0.0317 - accuracy
Epoch 2/10
20747/20747 [==============================] - 31s 1ms/step - loss: 0.0207 - accuracy
Epoch 3/10
20747/20747 [==============================] - 31s 1ms/step - loss: 0.0197 - accuracy
Epoch 4/10
20747/20747 [==============================] - 31s 1ms/step - loss: 0.0191 - accuracy
Epoch 5/10
20747/20747 [==============================] - 30s 1ms/step - loss: 0.0180 - accuracy
Epoch 6/10
20747/20747 [==============================] - 30s 1ms/step - loss: 0.0171 - accuracy
Epoch 7/10
20747/20747 [==============================] - 30s 1ms/step - loss: 0.0167 - accuracy
Epoch 8/10
20747/20747 [==============================] - 30s 1ms/step - loss: 0.0164 - accuracy
Epoch 9/10
20747/20747 [==============================] - 30s 1ms/step - loss: 0.0162 - accuracy
Epoch 10/10
20747/20747 [==============================] - 30s 1ms/step - loss: 0.0162 - accuracy
Score for fold 2: loss of 0.015777187421917915; accuracy of 99.60475564002991%
```

```
------------------------------------------------------------------------
Training for fold 3 ...
Epoch 1/10
20747/20747 [==============================] - 31s 1ms/step - loss: 0.0321 - accuracy
Epoch 2/10
20747/20747 [==============================] - 30s 1ms/step - loss: 0.0212 - accuracy
Epoch 3/10
20747/20747 [==============================] - 30s 1ms/step - loss: 0.0197 - accuracy
Epoch 4/10
20747/20747 [==============================] - 30s 1ms/step - loss: 0.0193 - accuracy
```