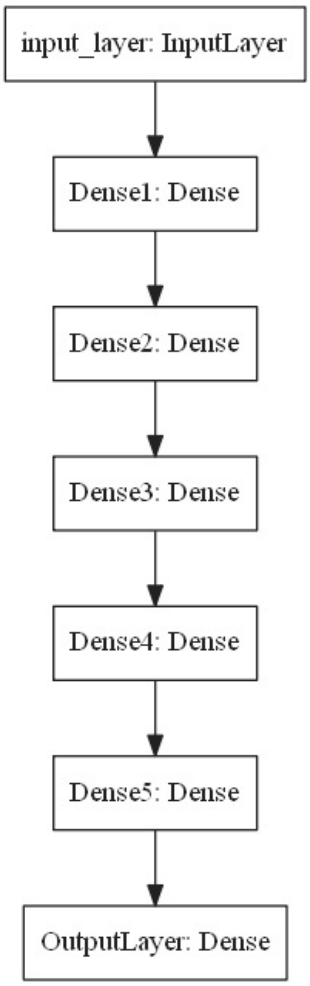
1. Download the data from here. You have to use data.csv file for this assignment

2. Code the model to classify data like below image. You can use any number of units in your Dense layers.



3. Writing Callbacks

You have to implement the following callbacks

- Write your own callback function, that has to print the micro F1 score and AUC score after each epoch.Do not use tf.keras.metrics for calculating AUC and F1 score.
- Save your model at every epoch if your validation accuracy is improved from previous epoch.
- You have to decay learning based on below conditions

```
  Cond1. If your validation accuracy at that epoch is less than previous epoch accuracy, you have to
decrese the
         learning rate by 10%.
  Cond2. For every 3rd epoch, decay your learning rate by 5%.
```

- If you are getting any NaN values(either weigths or loss) while training, you have to terminate your training.
- You have to stop the training if your validation accuracy is not increased in last 2 epochs.
- Use tensorboard for every model and analyse your scalar plots and histograms. (you need to upload the screenshots and write the observations for each model for evaluation)

In [1]:
```
# !gdown 15dCNcmKskcFVjs7R0ElQkR61Ex53uJpM
```

In [2]:
```
# Clear any logs from previous runs
!rm -rf ./N_logs/
```

In [3]:
```
import datetime
```

```
import numpy as np
import pandas as pd
# import tensorflow as tf

from sklearn.metrics import f1_score
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import train_test_split


from tensorflow.keras import optimizers
from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Activation

from tensorflow.keras.initializers import HeUniform
from tensorflow.keras.initializers import GlorotNormal
from tensorflow.keras.initializers import RandomUniform

from tensorflow.keras.callbacks import Callback
from tensorflow.keras.callbacks import TensorBoard
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.callbacks import ReduceLROnPlateau

%load_ext tensorboard
```

```
2022-03-05 16:19:30.742067: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynami
c library 'libcudart.so.11.0'; dlerror: libcudart.so.11.0: cannot open shared object file: No such file or direct
ory
2022-03-05 16:19:30.742121: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if y
ou do not have a GPU set up on your machine.
```

In [4]:

```
# https://www.tensorflow.org/tutorials/load_data/csv

data = pd.read_csv('data.csv')

print(f'Shape of data set : {data.shape}')
# pd.read_csv('data.csv').head()
```

```
Shape of data set : (20000, 3)
```

In [5]:

```
# Checking how the data is balanced !

data.label.value_counts()
```

Out[5]:

```
0.0    10000
1.0    10000
Name: label, dtype: int64
```

In [6]:

```
# Splitting data into train-test

y_label = data.label
x_values = data.drop(['label'], axis = 1)

x_train, x_test, y_train, y_test = train_test_split(x_values, y_label, test_size = 0.3, random_state = 5)
```

Custom Callbacks

In [7]:

```
class roc_score(Callback):

    def __init__(self, train_x, test_x, train_y, test_y):

        self.x_tr = train_x
        self.y_tr = train_y
        self.x_te = test_x
        self.y_te = test_y

    def on_epoch_end(self, epoch, logs = {}):

        y_pred_tr = self.model.predict(self.x_tr)
        y_pred_te = self.model.predict(self.x_te)

        roc_tr = roc_auc_score(self.y_tr, y_pred_tr)
        roc_te = roc_auc_score(self.y_te, y_pred_te)
```

```
        print(f' ROC_test:{round(roc_te, 3)}')

class f1_score_(Callback):

    def __init__(self, train_x, test_x, train_y, test_y):

        self.x_tr = train_x
        self.y_tr = train_y
        self.x_te = test_x
        self.y_te = test_y

    def on_epoch_end(self, epoch, logs = {}):

        y_pred_tr = (np.array(self.model.predict(self.x_tr))).round()
        y_pred_te = (np.array(self.model.predict(self.x_te))).round()

        f1_tr = f1_score(self.y_tr, y_pred_tr, average = 'micro')
        f1_te = f1_score(self.y_te, y_pred_te, average = 'micro')

        print(f' MicroF1_Test:{round(f1_te, 3)}')


class TerminateNaN(Callback):

    def on_epoch_end(self, epoch, logs = {}):

        loss = logs.get('loss')

        if loss is not None:

            if np.isnan(loss) or np.isinf(loss):
                print(F'Invalid loss and terminated at epoch {epoch}')
                self.model.stop_training = True


roc_score = roc_score(x_train, x_test, y_train, y_test)

f1_score_ = f1_score_(x_train, x_test, y_train, y_test)

terminateNaN = TerminateNaN()
```

In [8]:

```
# https://keras.io/api/callbacks/model_checkpoint/
# https://keras.io/api/callbacks/early_stopping/
# https://keras.io/api/callbacks/reduce_lr_on_plateau/

early_stopping = EarlyStopping(monitor = 'val_accuracy', patience = 2, verbose = 1)

filpath = 'me_model_save/epo_{epoch:02d}-accu_{val_accuracy:.4f}.hdf5'
model_check_point = ModelCheckpoint(filpath, monitor = 'val_accuracy', patience = 1, verbose = 1)


# factor: factor by which the learning rate will be reduced. new_lr = lr * factor
# lr = 0.1, factor = 0.9.
# Then, new_lr = 0.09 ==> 10% reduction

epoch_1 = ReduceLROnPlateau(monitor = 'val_accuracy', factor = 0.9, verbose = 1, patience = 1) # 10% reduction
epoch_3 = ReduceLROnPlateau(monitor = 'val_accuracy', factor = 0.95, verbose = 1, patience = 3) # 5% reduction
```

In [9]:

```
def create_model(activ, initializer):

    model = Sequential()
    model.add(Dense(32, input_shape = (2,), kernel_initializer = initializer, name = 'input_layer'))
    model.add(Dense(32, activation = activ, kernel_initializer = initializer, name = 'dense1'))
    model.add(Dense(32, activation = activ, kernel_initializer = initializer, name = 'dense2'))
    model.add(Dense(32, activation = activ, kernel_initializer = initializer, name = 'dense3'))
    model.add(Dense(32, activation = activ, kernel_initializer = initializer, name = 'dense4'))
    model.add(Dense(32, activation = activ, kernel_initializer = initializer, name = 'dense5'))
    model.add(Dense(1, activation = 'sigmoid', name = 'output_layer'))

    model.summary()

    return model

EPOCH = 20
```

Model-1

1. Use tanh as an activation for every layer except output layer.
2. use SGD with momentum as optimizer.
3. use RandomUniform(0,1) as initilizer.
3. Analyze your output and training process.

In [10]:

```
kernel_init = RandomUniform(0, 1)

model = create_model('tanh', kernel_init)

optimizer_ = optimizers.SGD(learning_rate = 0.001, momentum = 0.9, name = 'SGD')

model.compile(optimizer = optimizer_, loss = 'binary_crossentropy', metrics = ['accuracy'])
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_layer (Dense)         (None, 32)                96

 dense1 (Dense)              (None, 32)                1056

 dense2 (Dense)              (None, 32)                1056

 dense3 (Dense)              (None, 32)                1056

 dense4 (Dense)              (None, 32)                1056

 dense5 (Dense)              (None, 32)                1056

 output_layer (Dense)        (None, 1)                 33

=================================================================
Total params: 5,409
Trainable params: 5,409
Non-trainable params: 0
_________________________________________________________________
2022-03-05 16:19:32.506025: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynami
c library 'libcuda.so.1'; dlerror: libcuda.so.1: cannot open shared object file: No such file or directory
2022-03-05 16:19:32.506055: W tensorflow/stream_executor/cuda/cuda_driver.cc:269] failed call to cuInit: UNKNOWN
ERROR (303)
2022-03-05 16:19:32.506072: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:156] kernel driver does not app
ear to be running on this host (ubuntu): /proc/driver/nvidia/version does not exist
2022-03-05 16:19:32.506732: I tensorflow/core/platform/cpu_feature_guard.cc:151] This TensorFlow binary is optimi
zed with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critica
l operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
```

In [11]:

```
# TensorBoard

log_dir = 'N_logs/1_tanh_Random'
tensorboard_ = TensorBoard(log_dir = log_dir, histogram_freq = 1, write_graph = True)

CallBack_list = [roc_score, f1_score_, terminateNaN, early_stopping, model_check_point,
                                                                      epoch_1, epoch_3, tensorboard_]
```

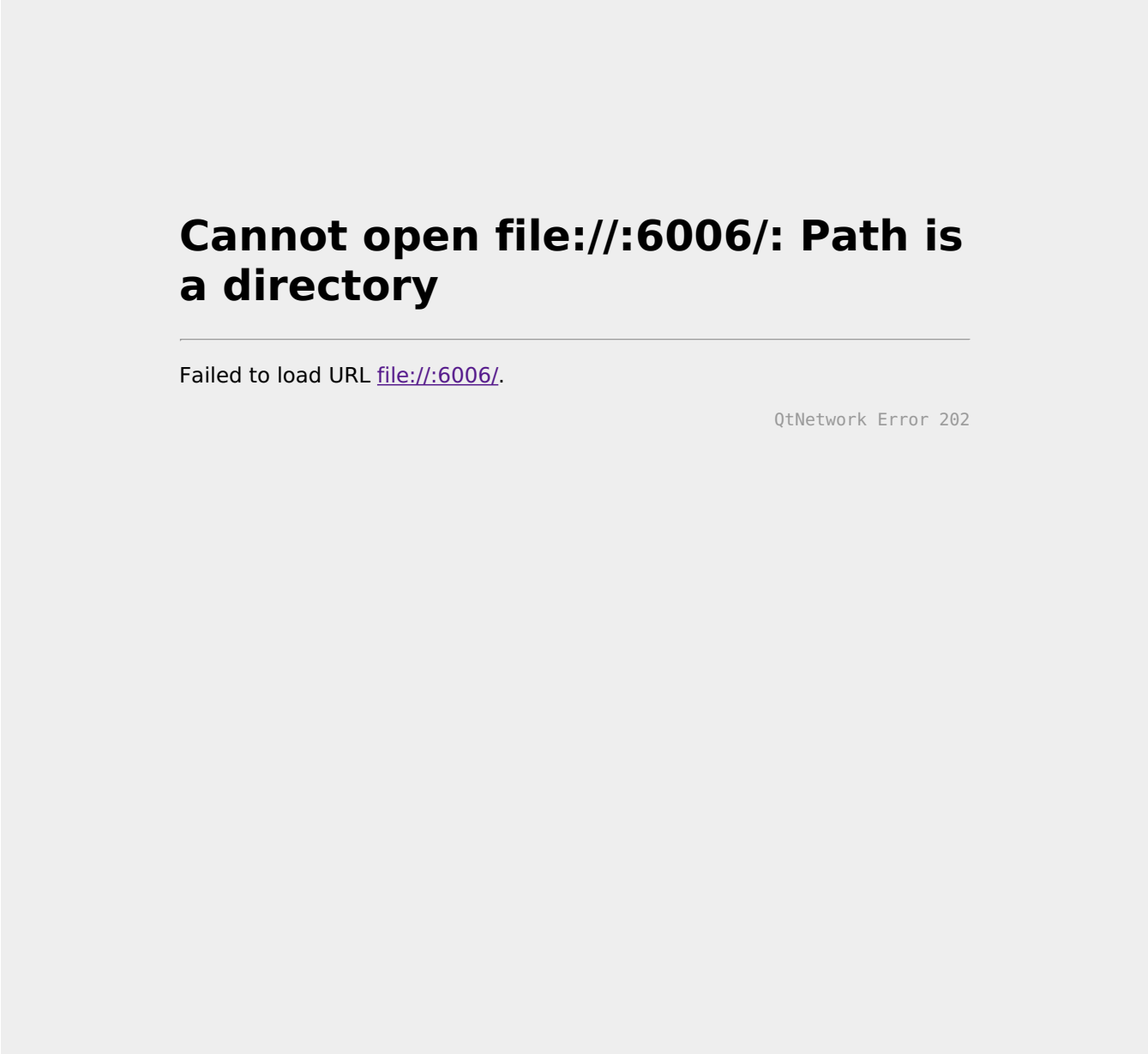
In [12]:

```
history = model.fit(x = x_train, y = y_train, epochs = EPOCH, verbose = 1,
            validation_data = (x_test, y_test), callbacks = CallBack_list)
```

```
Epoch 1/20
  1/438 [..............................] - ETA: 3:33 - loss: 0.6914 - accuracy: 0.5312WARNING:tensorflow:Callback
method `on_train_batch_end` is slow compared to the batch time (batch time: 0.0017s vs `on_train_batch_end` time:
0.0028s). Check your callbacks.
425/438 [============================>.] - ETA: 0s - loss: 0.6936 - accuracy: 0.4993 ROC_test:0.496
 MicroF1_Test:0.496

Epoch 1: saving model to me_model_save/epo_01-accu_0.4960.hdf5
438/438 [==============================] - 3s 6ms/step - loss: 0.6935 - accuracy: 0.4995 - val_loss: 0.6936 - val
_accuracy: 0.4960 - lr: 0.0010
Epoch 2/20
429/438 [============================>.] - ETA: 0s - loss: 0.6939 - accuracy: 0.4927 ROC_test:0.496
 MicroF1_Test:0.496

Epoch 2: saving model to me_model_save/epo_02-accu_0.4960.hdf5

Epoch 2: ReduceLROnPlateau reducing learning rate to 0.0009000000427477062.
438/438 [==============================] - 3s 6ms/step - loss: 0.6939 - accuracy: 0.4918 - val_loss: 0.6933 - val
_accuracy: 0.4960 - lr: 9.0000e-04
Epoch 3/20
414/438 [===========================>..] - ETA: 0s - loss: 0.6937 - accuracy: 0.4973 ROC_test:0.504
 MicroF1_Test:0.504

Epoch 3: saving model to me_model_save/epo_03-accu_0.5040.hdf5
438/438 [==============================] - 2s 5ms/step - loss: 0.6937 - accuracy: 0.4975 - val_loss: 0.6932 - val
_accuracy: 0.5040 - lr: 9.0000e-04
Epoch 4/20
417/438 [===========================>..] - ETA: 0s - loss: 0.6939 - accuracy: 0.5017 ROC_test:0.504
```

```
 MicroF1_Test:0.504

Epoch 4: saving model to me_model_save/epo_04-accu_0.5040.hdf5

Epoch 4: ReduceLROnPlateau reducing learning rate to 0.0008100000384729356.
438/438 [==============================] - 2s 5ms/step - loss: 0.6939 - accuracy: 0.5019 - val_loss: 0.6934 - val
_accuracy: 0.5040 - lr: 8.1000e-04
Epoch 5/20
431/438 [============================>.] - ETA: 0s - loss: 0.6933 - accuracy: 0.5081 ROC_test:0.504
 MicroF1_Test:0.494

Epoch 5: saving model to me_model_save/epo_05-accu_0.4942.hdf5

Epoch 5: ReduceLROnPlateau reducing learning rate to 0.0007290000503417104.
438/438 [==============================] - 2s 6ms/step - loss: 0.6933 - accuracy: 0.5073 - val_loss: 0.6932 - val
_accuracy: 0.4942 - lr: 7.2900e-04
Epoch 5: early stopping
```

In [13]:
```
%tensorboard --logdir N_logs/
```



Model-2

1. Use relu as an activation for every layer except output layer.
2. use SGD with momentum as optimizer.
3. use RandomUniform(0,1) as initilizer.
3. Analyze your output and training process.

In [14]:
```
kernel_init = RandomUniform(0, 1)
```

```
model = create_model('relu', kernel_init)

optimizer_ = optimizers.SGD(learning_rate = 0.001, momentum = 0.9, name = 'SGD')

model.compile(optimizer = optimizer_, loss = 'binary_crossentropy', metrics = ['accuracy'])
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_layer (Dense)         (None, 32)                96

 dense1 (Dense)              (None, 32)                1056

 dense2 (Dense)              (None, 32)                1056

 dense3 (Dense)              (None, 32)                1056

 dense4 (Dense)              (None, 32)                1056

 dense5 (Dense)              (None, 32)                1056

 output_layer (Dense)        (None, 1)                 33

=================================================================
Total params: 5,409
Trainable params: 5,409
Non-trainable params: 0
_________________________________________________________________
```

In [15]:

```
# TensorBoard

log_dir = 'N_logs/2_relu_Random'
tensorboard_ = TensorBoard(log_dir = log_dir, histogram_freq = 1, write_graph = True)

CallBack_list = [roc_score, f1_score_, terminateNaN, early_stopping, model_check_point,
                                                                      epoch_1, epoch_3, tensorboard_]
```

In [16]:

```
history = model.fit(x = x_train, y = y_train, epochs = EPOCH, verbose = 1,
            validation_data = (x_test, y_test), callbacks = CallBack_list)
```

```
Epoch 1/20
  1/438 [..............................] - ETA: 2:46 - loss: 174661.0469 - accuracy: 0.5312WARNING:tensorflow:Cal
lback method `on_train_batch_end` is slow compared to the batch time (batch time: 0.0027s vs `on_train_batch_end`
time: 0.0027s). Check your callbacks.
432/438 [============================>.] - ETA: 0s - loss: 1213.5258 - accuracy: 0.4941 ROC_test:0.5
 MicroF1_Test:0.494

Epoch 1: saving model to me_model_save/epo_01-accu_0.4942.hdf5
438/438 [==============================] - 3s 7ms/step - loss: 1198.2787 - accuracy: 0.4941 - val_loss: 0.6932 -
val_accuracy: 0.4942 - lr: 0.0010
Epoch 2/20
421/438 [===========================>..] - ETA: 0s - loss: 0.6932 - accuracy: 0.5029 ROC_test:0.5
 MicroF1_Test:0.494

Epoch 2: saving model to me_model_save/epo_02-accu_0.4942.hdf5

Epoch 2: ReduceLROnPlateau reducing learning rate to 0.0009900000427477062.
438/438 [==============================] - 2s 5ms/step - loss: 0.6932 - accuracy: 0.5025 - val_loss: 0.6932 - val
_accuracy: 0.4942 - lr: 9.0000e-04
Epoch 3/20
430/438 [============================>.] - ETA: 0s - loss: 0.6932 - accuracy: 0.5008 ROC_test:0.5
 MicroF1_Test:0.494

Epoch 3: saving model to me_model_save/epo_03-accu_0.4942.hdf5

Epoch 3: ReduceLROnPlateau reducing learning rate to 0.0008100000384729356.
438/438 [==============================] - 2s 5ms/step - loss: 0.6932 - accuracy: 0.5006 - val_loss: 0.6932 - val
_accuracy: 0.4942 - lr: 8.1000e-04
Epoch 3: early stopping
```

In [17]:

```
%tensorboard --logdir N_logs/
```

```
Reusing TensorBoard on port 6006 (pid 3916), started 0:00:09 ago. (Use '!kill 3916' to kill it.)
```

Cannot open file://:6006/: Path is a directory

Failed to load URL file://:6006/.

QtNetwork Error 202

Model-3

1. Use relu as an activation for every layer except output layer.
2. use SGD with momentum as optimizer.
3. use he_uniform() as initilizer.
3. Analyze your output and training process.

In [18]:

```
kernel_init = HeUniform()

model = create_model('relu', kernel_init)

optimizer_ = optimizers.SGD(learning_rate = 0.001, momentum = 0.9, name = 'SGD')

model.compile(optimizer = optimizer_, loss = 'binary_crossentropy', metrics = ['accuracy'])
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_layer (Dense)         (None, 32)                96

 dense1 (Dense)              (None, 32)                1056

 dense2 (Dense)              (None, 32)                1056

 dense3 (Dense)              (None, 32)                1056

 dense4 (Dense)              (None, 32)                1056

 dense5 (Dense)              (None, 32)                1056

 output_layer (Dense)        (None, 1)                 33

=================================================================
Total params: 5,409
Trainable params: 5,409
Non-trainable params: 0
```

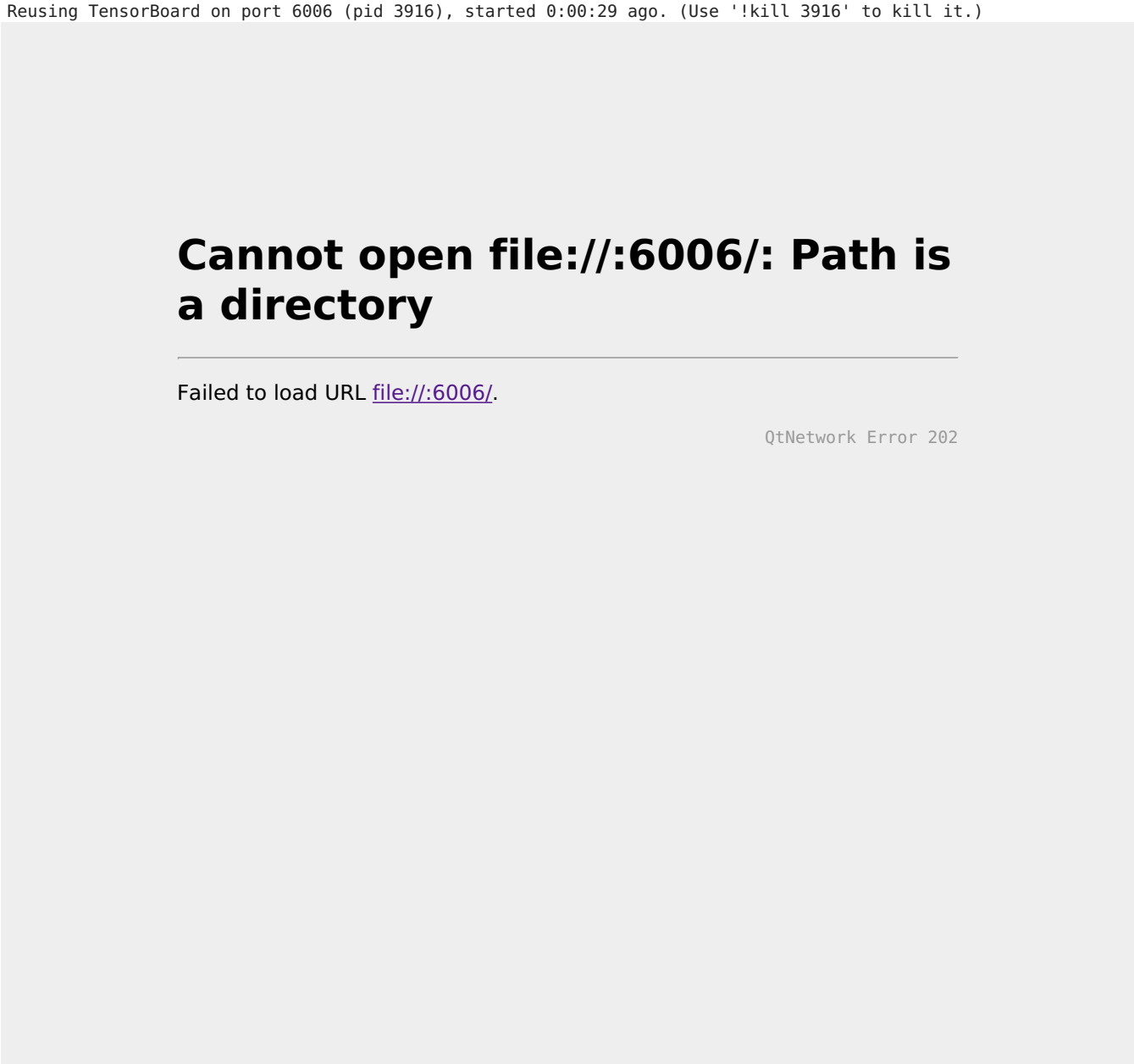
In [19]:
```
# TensorBoard

log_dir = 'N_logs/3_relu_HeUniform'
tensorboard_ = TensorBoard(log_dir = log_dir, histogram_freq = 1, write_graph = True)

CallBack_list = [roc_score, f1_score_, terminateNaN, early_stopping, model_check_point,
                                                                      epoch_1, epoch_3, tensorboard_]
```

In [20]:
```
history = model.fit(x = x_train, y = y_train, epochs = EPOCH, verbose = 1,
            validation_data = (x_test, y_test), callbacks = CallBack_list)
```

```
Epoch 1/20
  1/438 [..............................] - ETA: 3:18 - loss: 0.9620 - accuracy: 0.5312WARNING:tensorflow:Callback
method `on_train_batch_end` is slow compared to the batch time (batch time: 0.0015s vs `on_train_batch_end` time:
0.0020s). Check your callbacks.
415/438 [===========================>..] - ETA: 0s - loss: 0.6914 - accuracy: 0.5336 ROC_test:0.705
 MicroF1_Test:0.574

Epoch 1: saving model to me_model_save/epo_01-accu_0.5740.hdf5
438/438 [==============================] - 3s 6ms/step - loss: 0.6904 - accuracy: 0.5369 - val_loss: 0.6698 - val
_accuracy: 0.5740 - lr: 0.0010
Epoch 2/20
438/438 [==============================] - ETA: 0s - loss: 0.6576 - accuracy: 0.6280 ROC_test:0.721
 MicroF1_Test:0.66

Epoch 2: saving model to me_model_save/epo_02-accu_0.6602.hdf5
438/438 [==============================] - 2s 5ms/step - loss: 0.6576 - accuracy: 0.6280 - val_loss: 0.6470 - val
_accuracy: 0.6602 - lr: 0.0010
Epoch 3/20
417/438 [===========================>..] - ETA: 0s - loss: 0.6328 - accuracy: 0.6586 ROC_test:0.72
 MicroF1_Test:0.646

Epoch 3: saving model to me_model_save/epo_03-accu_0.6462.hdf5

Epoch 3: ReduceLROnPlateau reducing learning rate to 0.0009000000427477062.
438/438 [==============================] - 2s 6ms/step - loss: 0.6322 - accuracy: 0.6585 - val_loss: 0.6305 - val
_accuracy: 0.6462 - lr: 9.0000e-04
Epoch 4/20
411/438 [===========================>..] - ETA: 0s - loss: 0.6162 - accuracy: 0.6635 ROC_test:0.73
 MicroF1_Test:0.663

Epoch 4: saving model to me_model_save/epo_04-accu_0.6627.hdf5
438/438 [==============================] - 2s 5ms/step - loss: 0.6155 - accuracy: 0.6627 - val_loss: 0.6132 - val
_accuracy: 0.6627 - lr: 9.0000e-04
Epoch 5/20
433/438 [============================>.] - ETA: 0s - loss: 0.6072 - accuracy: 0.6677 ROC_test:0.729
 MicroF1_Test:0.659

Epoch 5: saving model to me_model_save/epo_05-accu_0.6587.hdf5

Epoch 5: ReduceLROnPlateau reducing learning rate to 0.0008100000384729356.
438/438 [==============================] - 2s 6ms/step - loss: 0.6071 - accuracy: 0.6674 - val_loss: 0.6135 - val
_accuracy: 0.6587 - lr: 8.1000e-04
Epoch 6/20
431/438 [============================>.] - ETA: 0s - loss: 0.6024 - accuracy: 0.6724 ROC_test:0.732
 MicroF1_Test:0.667

Epoch 6: saving model to me_model_save/epo_06-accu_0.6668.hdf5
438/438 [==============================] - 2s 5ms/step - loss: 0.6031 - accuracy: 0.6719 - val_loss: 0.6054 - val
_accuracy: 0.6668 - lr: 8.1000e-04
Epoch 7/20
433/438 [============================>.] - ETA: 0s - loss: 0.6035 - accuracy: 0.6693 ROC_test:0.73
 MicroF1_Test:0.664

Epoch 7: saving model to me_model_save/epo_07-accu_0.6643.hdf5

Epoch 7: ReduceLROnPlateau reducing learning rate to 0.0007290000503417104.
438/438 [==============================] - 2s 5ms/step - loss: 0.6027 - accuracy: 0.6704 - val_loss: 0.6085 - val
_accuracy: 0.6643 - lr: 7.2900e-04
Epoch 8/20
432/438 [============================>.] - ETA: 0s - loss: 0.6011 - accuracy: 0.6716 ROC_test:0.727
 MicroF1_Test:0.661

Epoch 8: saving model to me_model_save/epo_08-accu_0.6612.hdf5

Epoch 8: ReduceLROnPlateau reducing learning rate to 0.0006561000715009868.
438/438 [==============================] - 3s 6ms/step - loss: 0.6010 - accuracy: 0.6721 - val_loss: 0.6111 - val
_accuracy: 0.6612 - lr: 6.5610e-04
Epoch 8: early stopping
```

In [21]:
```
%tensorboard --logdir N_logs/
```

```
Reusing TensorBoard on port 6006 (pid 3916), started 0:00:29 ago. (Use '!kill 3916' to kill it.)
```



Model-4

1. Try with any values to get better accuracy/f1 score.

In [22]:
```
kernel_init = GlorotNormal()

model = create_model('relu', kernel_init)

optimizer_ = optimizers.SGD(learning_rate = 0.001, momentum = 0.9, name = 'SGD')

model.compile(optimizer = optimizer_, loss = 'binary_crossentropy', metrics = ['accuracy'])
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_layer (Dense)         (None, 32)                96

 dense1 (Dense)              (None, 32)                1056

 dense2 (Dense)              (None, 32)                1056

 dense3 (Dense)              (None, 32)                1056

 dense4 (Dense)              (None, 32)                1056

 dense5 (Dense)              (None, 32)                1056
```

```
 output_layer (Dense)        (None, 1)                 33

=================================================================
Total params: 5,409
Trainable params: 5,409
Non-trainable params: 0
_________________________________________________________________
```

In [23]:

```
# TensorBoard

log_dir = 'N_logs/4_relu_GlorotNormal'
tensorboard_ = TensorBoard(log_dir = log_dir, histogram_freq = 1, write_graph = True)

CallBack_list = [roc_score, f1_score_, terminateNaN, early_stopping, model_check_point,
                                                                      epoch_1, epoch_3, tensorboard_]
```

In [24]:

```
history = model.fit(x = x_train, y = y_train, epochs = EPOCH, verbose = 1,
            validation_data = (x_test, y_test), callbacks = CallBack_list)
```

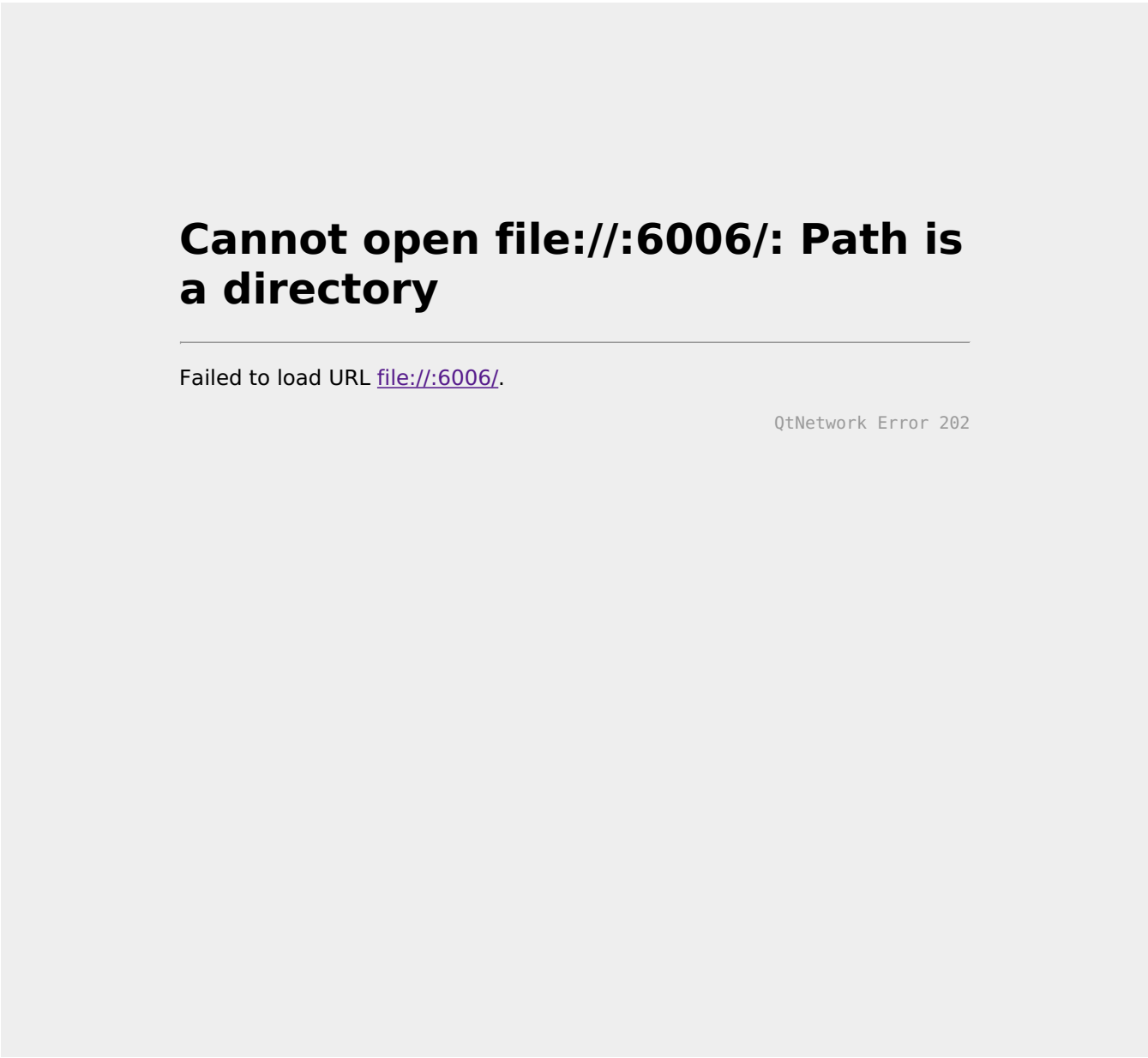
```
Epoch 1/20
  1/438 [..............................] - ETA: 2:38 - loss: 0.6879 - accuracy: 0.6875WARNING:tensorflow:Callback
method `on_train_batch_end` is slow compared to the batch time (batch time: 0.0014s vs `on_train_batch_end` time:
0.0021s). Check your callbacks.
424/438 [============================>.] - ETA: 0s - loss: 0.6912 - accuracy: 0.5394 ROC_test:0.619
 MicroF1_Test:0.579

Epoch 1: saving model to me_model_save/epo_01-accu_0.5787.hdf5
438/438 [==============================] - 3s 5ms/step - loss: 0.6912 - accuracy: 0.5402 - val_loss: 0.6889 - val
_accuracy: 0.5787 - lr: 0.0010
Epoch 2/20
428/438 [============================>.] - ETA: 0s - loss: 0.6864 - accuracy: 0.5845 ROC_test:0.652
 MicroF1_Test:0.607

Epoch 2: saving model to me_model_save/epo_02-accu_0.6072.hdf5
438/438 [==============================] - 2s 6ms/step - loss: 0.6864 - accuracy: 0.5843 - val_loss: 0.6830 - val
_accuracy: 0.6072 - lr: 0.0010
Epoch 3/20
415/438 [===========================>..] - ETA: 0s - loss: 0.6805 - accuracy: 0.6052 ROC_test:0.669
 MicroF1_Test:0.617

Epoch 3: saving model to me_model_save/epo_03-accu_0.6173.hdf5
438/438 [==============================] - 2s 6ms/step - loss: 0.6802 - accuracy: 0.6048 - val_loss: 0.6754 - val
_accuracy: 0.6173 - lr: 0.0010
Epoch 4/20
411/438 [===========================>..] - ETA: 0s - loss: 0.6705 - accuracy: 0.6243 ROC_test:0.698
 MicroF1_Test:0.646

Epoch 4: saving model to me_model_save/epo_04-accu_0.6457.hdf5
438/438 [==============================] - 2s 6ms/step - loss: 0.6701 - accuracy: 0.6239 - val_loss: 0.6622 - val
_accuracy: 0.6457 - lr: 0.0010
Epoch 5/20
429/438 [============================>.] - ETA: 0s - loss: 0.6531 - accuracy: 0.6560 ROC_test:0.728
 MicroF1_Test:0.666

Epoch 5: saving model to me_model_save/epo_05-accu_0.6665.hdf5
438/438 [==============================] - 2s 6ms/step - loss: 0.6528 - accuracy: 0.6561 - val_loss: 0.6417 - val
_accuracy: 0.6665 - lr: 0.0010
Epoch 6/20
409/438 [===========================>..] - ETA: 0s - loss: 0.6286 - accuracy: 0.6696 ROC_test:0.73
 MicroF1_Test:0.663

Epoch 6: saving model to me_model_save/epo_06-accu_0.6630.hdf5

Epoch 6: ReduceLROnPlateau reducing learning rate to 0.0009000000427477062.
438/438 [==============================] - 2s 6ms/step - loss: 0.6282 - accuracy: 0.6695 - val_loss: 0.6210 - val
_accuracy: 0.6630 - lr: 9.0000e-04
Epoch 7/20
427/438 [============================>.] - ETA: 0s - loss: 0.6101 - accuracy: 0.6749 ROC_test:0.73
 MicroF1_Test:0.667

Epoch 7: saving model to me_model_save/epo_07-accu_0.6667.hdf5
438/438 [==============================] - 2s 6ms/step - loss: 0.6106 - accuracy: 0.6744 - val_loss: 0.6107 - val
_accuracy: 0.6667 - lr: 9.0000e-04
Epoch 8/20
426/438 [============================>.] - ETA: 0s - loss: 0.6039 - accuracy: 0.6732 ROC_test:0.732
 MicroF1_Test:0.668

Epoch 8: saving model to me_model_save/epo_08-accu_0.6682.hdf5
438/438 [==============================] - 2s 6ms/step - loss: 0.6036 - accuracy: 0.6735 - val_loss: 0.6076 - val
_accuracy: 0.6682 - lr: 9.0000e-04
Epoch 9/20
429/438 [============================>.] - ETA: 0s - loss: 0.6024 - accuracy: 0.6739 ROC_test:0.732
 MicroF1_Test:0.665
```

```
Epoch 9: saving model to me_model_save/epo_09-accu_0.6653.hdf5

Epoch 9: ReduceLROnPlateau reducing learning rate to 0.0008100000384729356.
438/438 [==============================] - 2s 6ms/step - loss: 0.6013 - accuracy: 0.6746 - val_loss: 0.6070 - val
_accuracy: 0.6653 - lr: 8.1000e-04
Epoch 10/20
429/438 [============================>.] - ETA: 0s - loss: 0.6011 - accuracy: 0.6696 ROC_test:0.733
 MicroF1_Test:0.667

Epoch 10: saving model to me_model_save/epo_10-accu_0.6670.hdf5

Epoch 10: ReduceLROnPlateau reducing learning rate to 0.0007290000503417104.
438/438 [==============================] - 2s 6ms/step - loss: 0.6009 - accuracy: 0.6701 - val_loss: 0.6059 - val
_accuracy: 0.6670 - lr: 7.2900e-04
Epoch 10: early stopping
```

In [25]:

```
%tensorboard --logdir N_logs/
```

```
Reusing TensorBoard on port 6006 (pid 3916), started 0:00:54 ago. (Use '!kill 3916' to kill it.)
```



Note

Make sure that you are plotting tensorboard plots either in your notebook or you can try to create a pdf file with all the tensorboard screenshots.Please write your analysis of tensorboard results for each model.

Comparision

In [26]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ['Model', 'Activation Function', 'Kernal Initializer', 'No. of Epochs ran', 'Loss', 'Accuracy', '
x.add_row([1, 'tanh', 'RandomUniform', 5, 0.6933, 0.5081, 0.504])
x.add_row([2, 'relu', 'RandomUniform', 3, 0.6932, 0.5006, 0.5])
x.add_row([3, 'relu', 'HeUniform', 8, 0.6011, 0.6776, 0.727])
x.add_row([4, 'relu', 'GlarotNormal', 10, 0.6011, 0.6701, 0.73])

print(x)
```

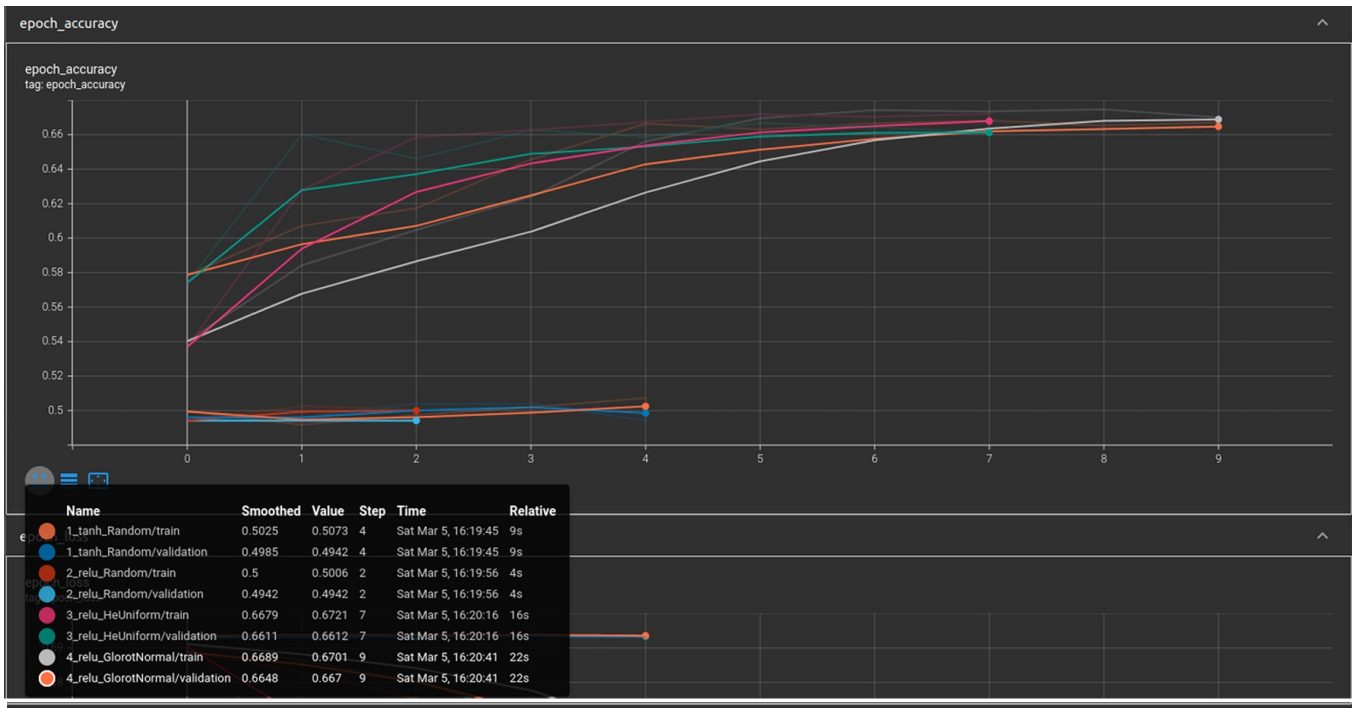
```
+-------+---------------------+--------------------+-------------------+--------+----------+----------+
| Model | Activation Function | Kernal Initializer | No. of Epochs ran |  Loss  | Accuracy | ROC Test |
+-------+---------------------+--------------------+-------------------+--------+----------+----------+
|   1   |         tanh        |   RandomUniform    |         5         | 0.6933 |  0.5081  |  0.504   |
|   2   |         relu        |   RandomUniform    |         3         | 0.6932 |  0.5006  |   0.5    |
|   3   |         relu        |     HeUniform      |         8         | 0.6011 |  0.6776  |  0.727   |
|   4   |         relu        |    GlarotNormal    |         10        | 0.6011 |  0.6701  |   0.73   |
+-------+---------------------+--------------------+-------------------+--------+----------+----------+
```

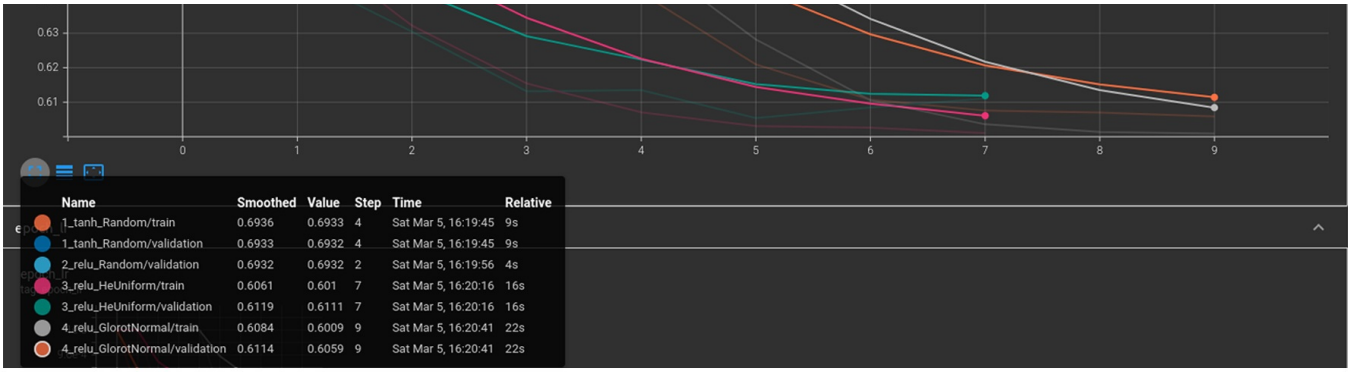
Observation

- This shows with activation function as `relu` and kernal initializer as `GlarotNormal` gives better **ROC Test** value than anu other models.
- Also this models runs for 10 epochs.

TensorBoard Outpots

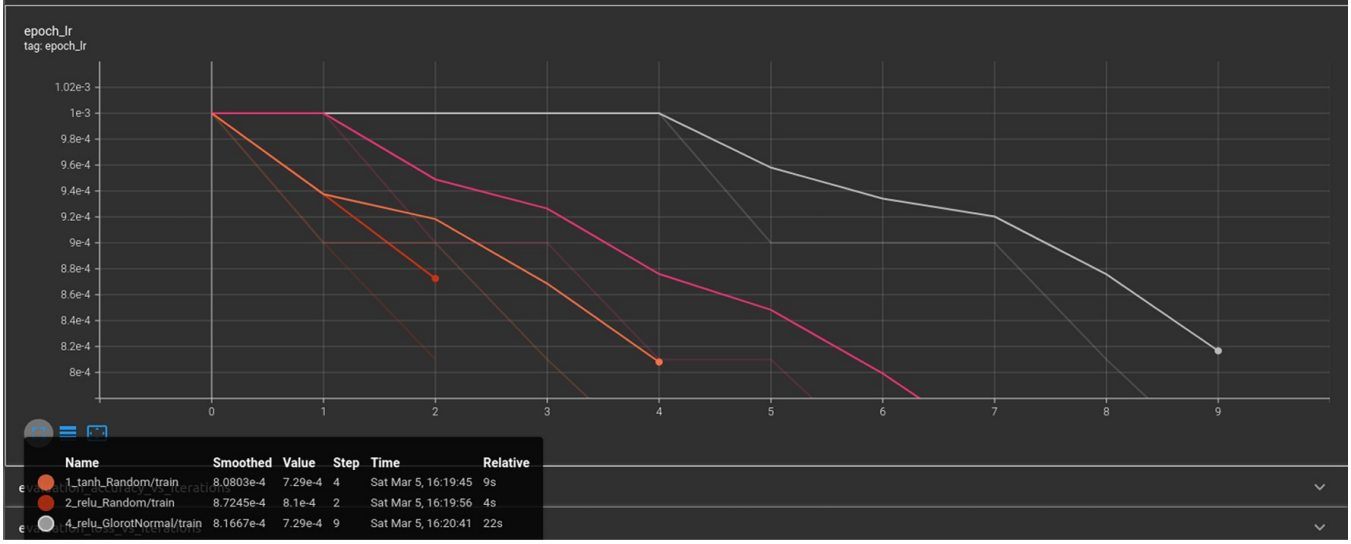
Scalar





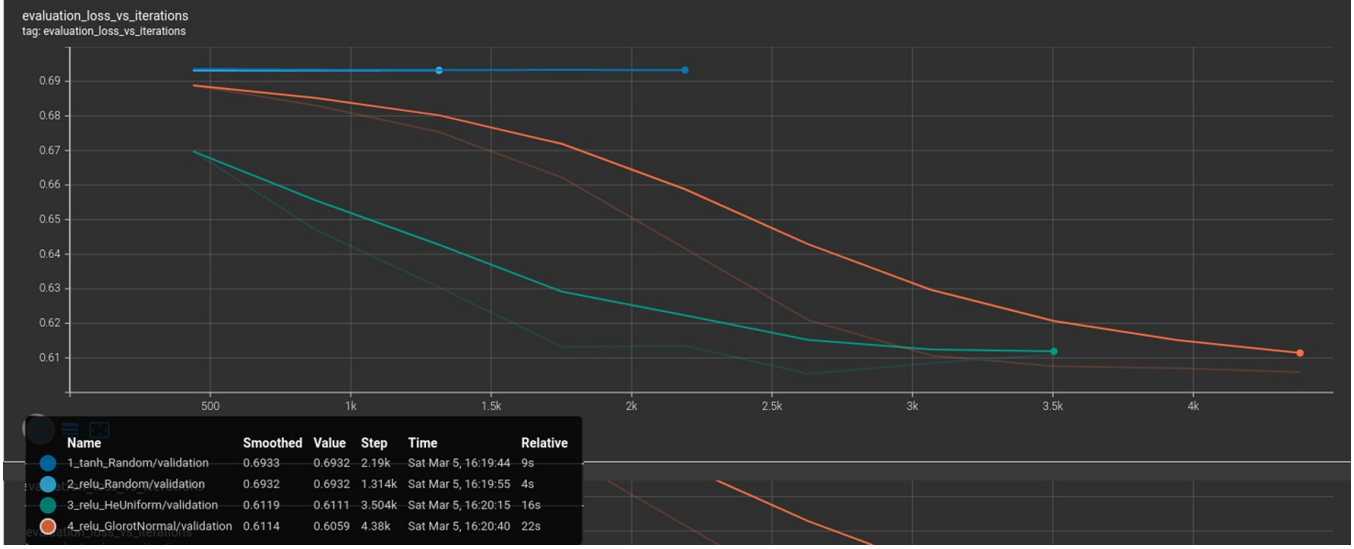
Name	Smoothed	Value	Step	Time	Relative
1_tanh_Random/train	0.6936	0.6933	4	Sat Mar 5, 16:19:45	9s
1_tanh_Random/validation	0.6933	0.6932	4	Sat Mar 5, 16:19:45	9s
2_relu_Random/validation	0.6932	0.6932	2	Sat Mar 5, 16:19:56	4s
3_relu_HeUniform/train	0.6061	0.601	7	Sat Mar 5, 16:20:16	16s
3_relu_HeUniform/validation	0.6119	0.6111	7	Sat Mar 5, 16:20:16	16s
4_relu_GlorotNormal/train	0.6084	0.6009	9	Sat Mar 5, 16:20:41	22s
4_relu_GlorotNormal/validation	0.6114	0.6059	9	Sat Mar 5, 16:20:41	22s

epoch_lr



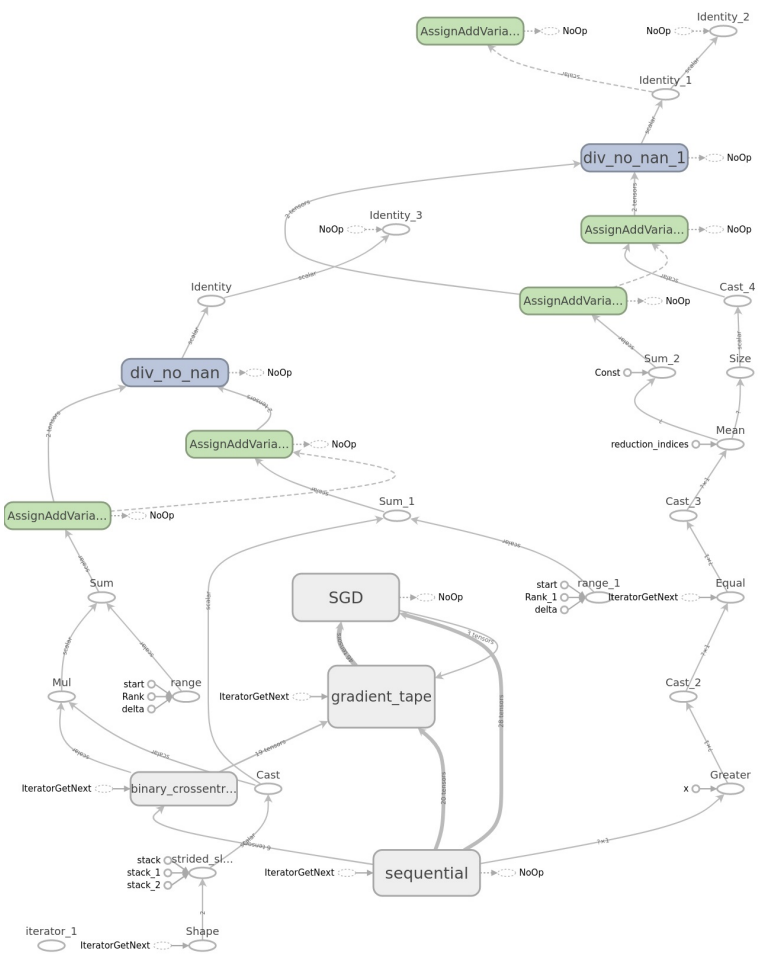
Name	Smoothed	Value	Step	Time	Relative
1_tanh_Random/train	8.0803e-4	7.29e-4	4	Sat Mar 5, 16:19:45	9s
2_relu_Random/train	8.7245e-4	8.1e-4	2	Sat Mar 5, 16:19:56	4s
4_relu_GlorotNormal/train	8.1667e-4	7.29e-4	9	Sat Mar 5, 16:20:41	22s

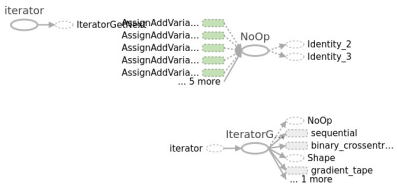
evaluation_loss_vs_iterations



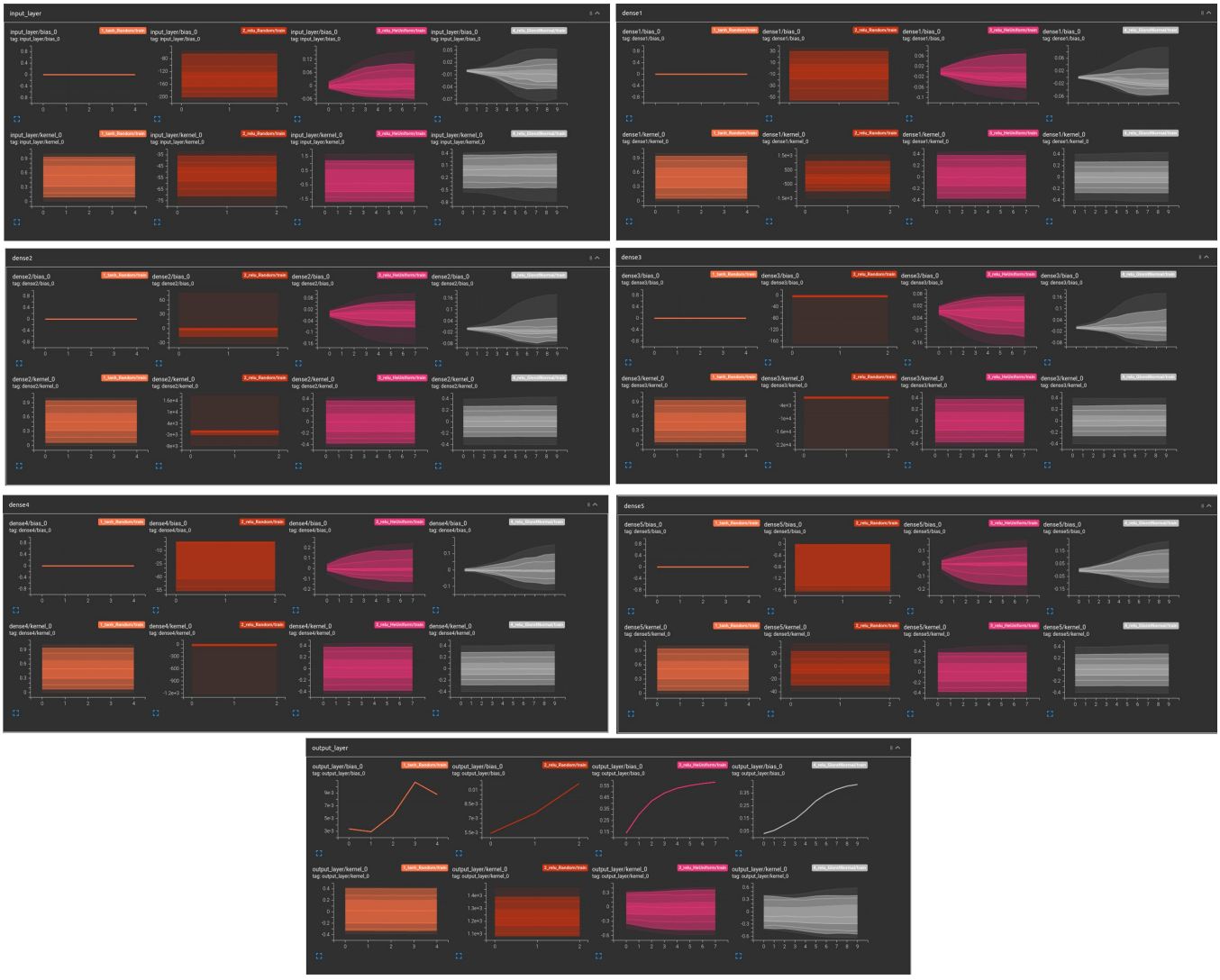
Name	Smoothed	Value	Step	Time	Relative
1_tanh_Random/validation	0.6933	0.6932	2.19k	Sat Mar 5, 16:19:44	9s
2_relu_Random/validation	0.6932	0.6932	1.314k	Sat Mar 5, 16:19:55	4s
3_relu_HeUniform/validation	0.6119	0.6111	3.504k	Sat Mar 5, 16:20:15	16s
4_relu_GlorotNormal/validation	0.6114	0.6059	4.38k	Sat Mar 5, 16:20:40	22s

Graphs

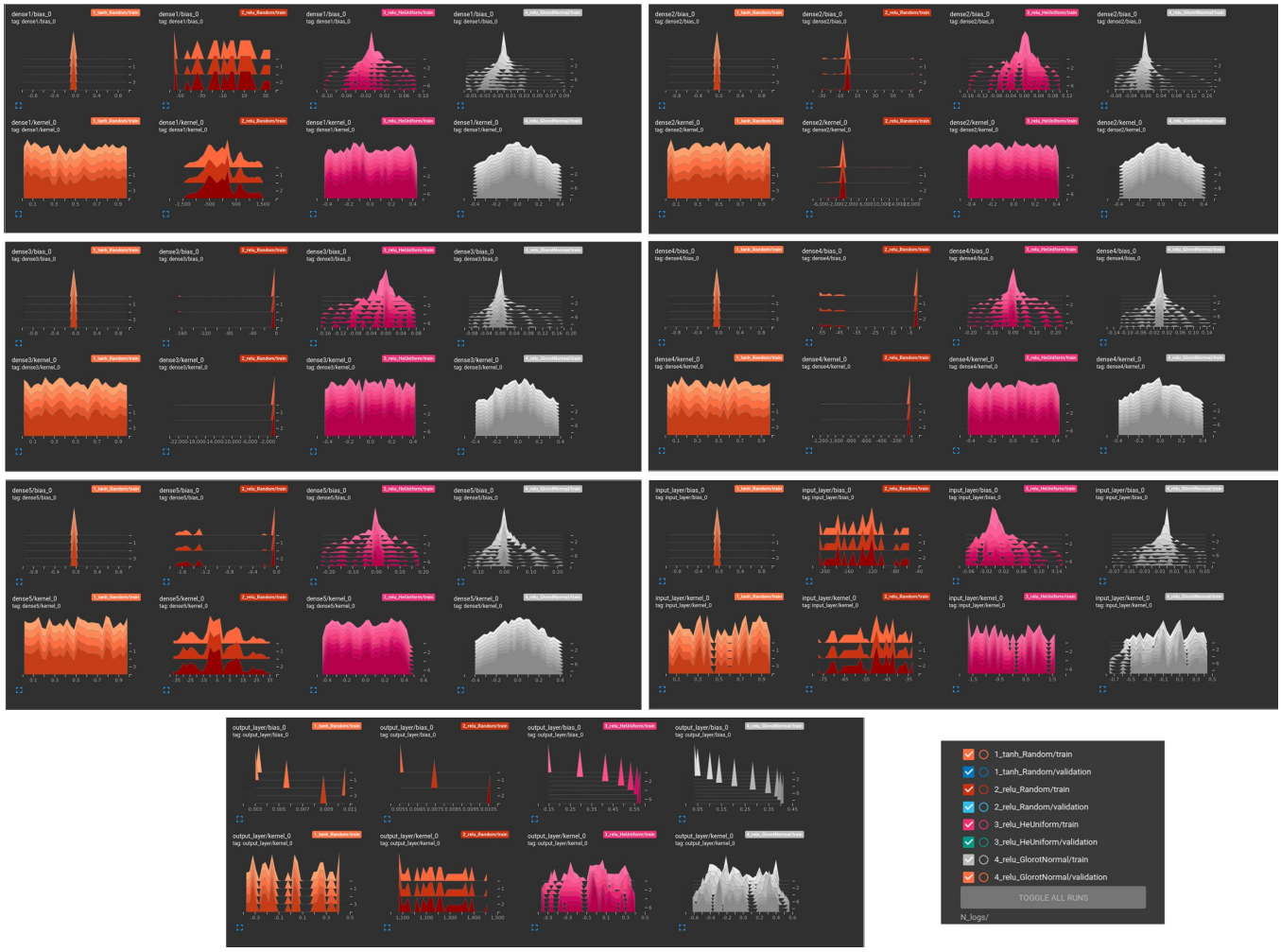




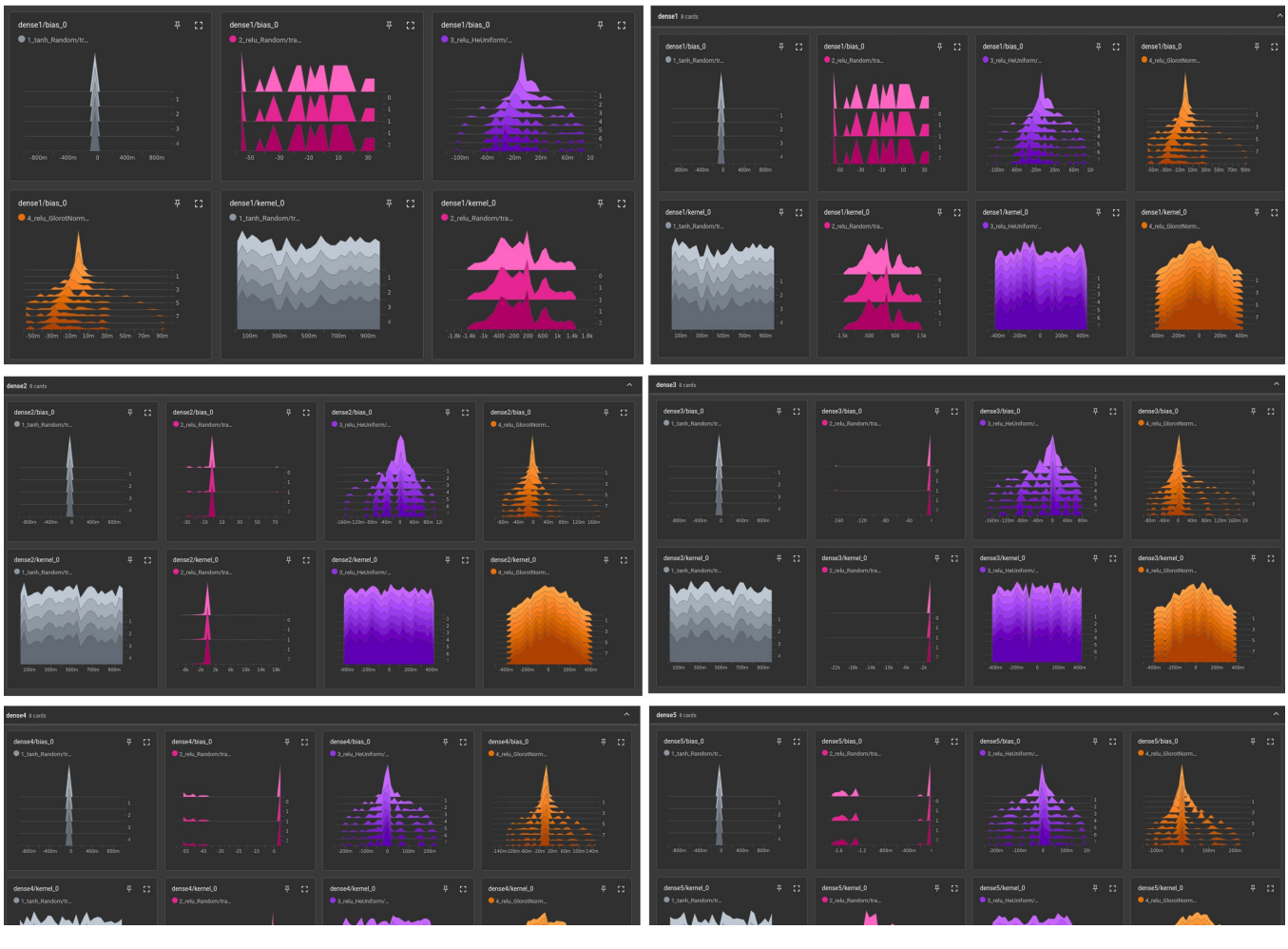
Distributions

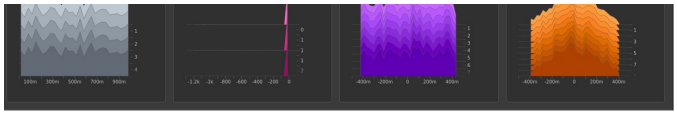

input_layer
dense1
dense2
dense3
dense4
dense5
output_layer

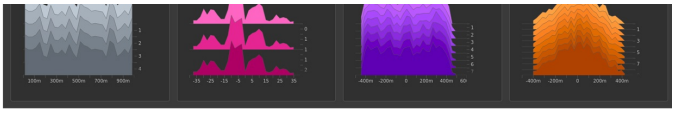
Histograms

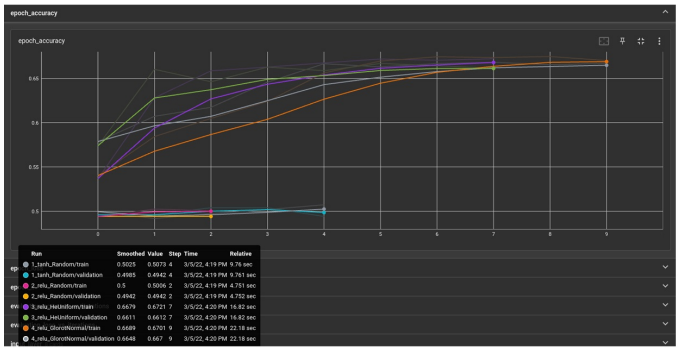

1_tanh_Random/train
1_tanh_Random/validation
2_relu_Random/train
2_relu_Random/validation
3_relu_HeUniform/train
3_relu_HeUniform/validation
4_relu_GlorotNormal/train
4_relu_GlorotNormal/validation
TOGGLE ALL RUNS
N_logs/

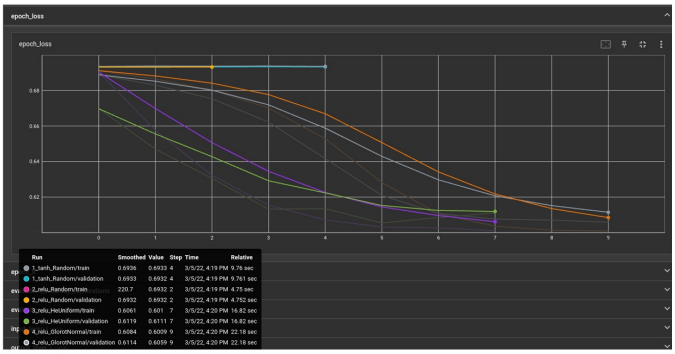
Time Series

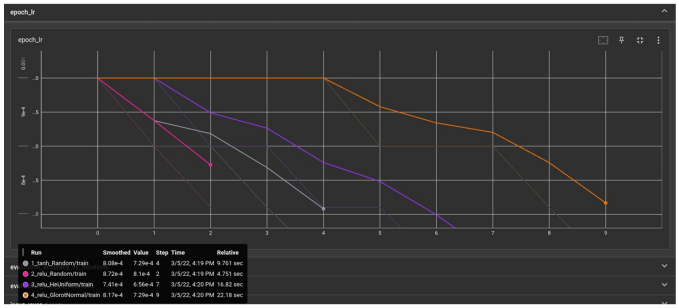


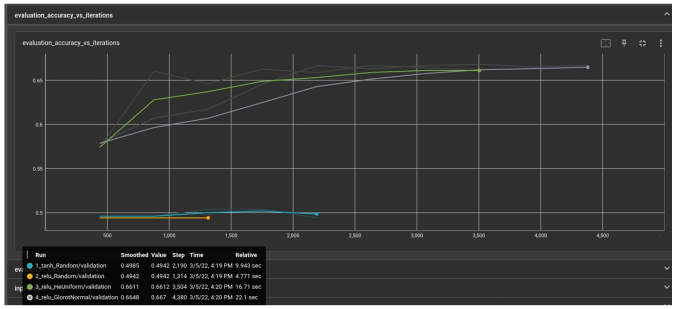


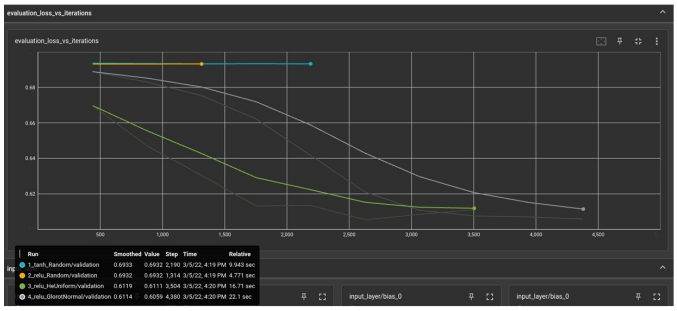


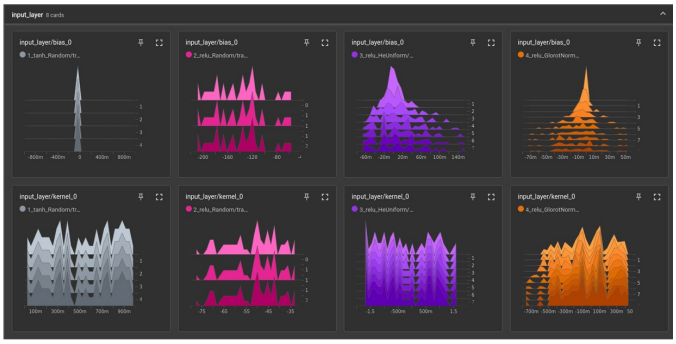

epoch_accuracy
epoch_accuracy
Run Smoothed Value Step Time Relative
1_tanh_Random/train 0.5025 0.5073 4 3/5/22, 4:19 PM 9.76 sec
1_tanh_Random/validation 0.4985 0.4942 4 3/5/22, 4:19 PM 9.761 sec
2_relu_Random/train 0.5 0.5006 2 3/5/22, 4:19 PM 4.751 sec
2_relu_Random/validation 0.4942 0.4942 2 3/5/22, 4:19 PM 4.752 sec
3_relu_HeUniform/train 0.6679 0.6721 7 3/5/22, 4:20 PM 16.82 sec
3_relu_HeUniform/validation 0.6611 0.6612 7 3/5/22, 4:20 PM 16.82 sec
4_relu_GlorotNormal/train 0.6689 0.6701 9 3/5/22, 4:20 PM 22.18 sec
4_relu_GlorotNormal/validation 0.6648 0.667 9 3/5/22, 4:20 PM 22.18 sec


epoch_loss
epoch_loss
Run Smoothed Value Step Time Relative
1_tanh_Random/train 0.6936 0.6933 4 3/5/22, 4:19 PM 9.76 sec
1_tanh_Random/validation 0.6933 0.6932 4 3/5/22, 4:19 PM 9.761 sec
2_relu_Random/train 220.7 0.6932 2 3/5/22, 4:19 PM 4.75 sec
2_relu_Random/validation 0.6932 0.6932 2 3/5/22, 4:19 PM 4.752 sec
3_relu_HeUniform/train 0.6061 0.601 7 3/5/22, 4:20 PM 16.82 sec
3_relu_HeUniform/validation 0.6119 0.6111 7 3/5/22, 4:20 PM 16.82 sec
4_relu_GlorotNormal/train 0.6084 0.6009 9 3/5/22, 4:20 PM 22.18 sec
4_relu_GlorotNormal/validation 0.6114 0.6059 9 3/5/22, 4:20 PM 22.18 sec

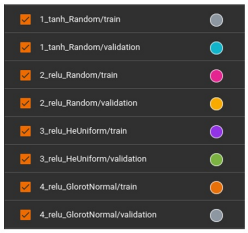

epoch_lr
epoch_lr
Run Smoothed Value Step Time Relative
1_tanh_Random/train 8.08e-4 7.29e-4 4 3/5/22, 4:19 PM 9.761 sec
2_relu_Random/train 8.72e-4 8.1e-4 2 3/5/22, 4:19 PM 4.751 sec
3_relu_HeUniform/train 7.41e-4 6.56e-4 7 3/5/22, 4:20 PM 16.82 sec
4_relu_GlorotNormal/train 8.17e-4 7.29e-4 9 3/5/22, 4:20 PM 22.18 sec


evaluation_accuracy_vs_iterations
evaluation_accuracy_vs_iterations
Run Smoothed Value Step Time Relative
1_tanh_Random/validation 0.4985 0.4942 2,190 3/5/22, 4:19 PM 9.943 sec
2_relu_Random/validation 0.4942 0.4942 1,314 3/5/22, 4:19 PM 4.771 sec
3_relu_HeUniform/validation 0.6611 0.6612 3,504 3/5/22, 4:20 PM 16.71 sec
4_relu_GlorotNormal/validation 0.6648 0.667 4,380 3/5/22, 4:20 PM 22.1 sec


evaluation_loss_vs_iterations
evaluation_loss_vs_iterations
Run Smoothed Value Step Time Relative
1_tanh_Random/validation 0.6933 0.6932 2,190 3/5/22, 4:19 PM 9.943 sec
2_relu_Random/validation 0.6932 0.6932 1,314 3/5/22, 4:19 PM 4.771 sec
3_relu_HeUniform/validation 0.6119 0.6111 3,504 3/5/22, 4:20 PM 16.71 sec
4_relu_GlorotNormal/validation 0.6114 0.6059 4,380 3/5/22, 4:20 PM 22.1 sec
input_layer/bias_0
input_layer/bias_0


input_layer 8 cards
input_layer/bias_0
1_tanh_Random/tr...
input_layer/bias_0
2_relu_Random/tra...
input_layer/bias_0
3_relu_HeUniform/...
input_layer/bias_0
4_relu_GlorotNorm...
input_layer/kernel_0
1_tanh_Random/tr...
input_layer/kernel_0
2_relu_Random/tra...
input_layer/kernel_0
3_relu_HeUniform/...
input_layer/kernel_0
4_relu_GlorotNorm...


output_layer 8 cards
output_layer/bias_0
1_tanh_Random/tr...
output_layer/bias_0
2_relu_Random/tra...
output_layer/bias_0
3_relu_HeUniform/...
output_layer/bias_0
4_relu_GlorotNorm...
output_layer/kernel_0
1_tanh_Random/tr...
output_layer/kernel_0
2_relu_Random/tra...
output_layer/kernel_0
3_relu_HeUniform/...
output_layer/kernel_0
4_relu_GlorotNorm...


1_tanh_Random/train
1_tanh_Random/validation
2_relu_Random/train
2_relu_Random/validation
3_relu_HeUniform/train
3_relu_HeUniform/validation
4_relu_GlorotNormal/train
4_relu_GlorotNormal/validation