Neural Networks - intro

Part 1 - XOR

1. Using the XOR dataset below, train (400 epochs) a neural network (NN) using 2, 3, 4, and 5 hidden layers (where each layer has only 2 neurons). For each n layers, store the resulting accuracy along with n. Plot the results to find what the optimal number of layers is.
2. Repeat the above with 3 neurons in each Hidden layers. How do these results compare to the 2 neuron layers?
3. Repeat the above with 4 neurons in each Hidden layers. How do these results compare to the 2 and 3 neuron layers?
4. Using the most optimal configuraion (n-layers, k-neurons per layer), compare how `tanh` , `sigmoid` , `softplus` and `relu` effect the loss after 400 epochs. Try other Activation functions as well (https://keras.io/activations/ (https://keras.io/activations/))
5. Again with the most optimal setup, try other optimizers (instead of `SGD`) and report on the loss score. (https://keras.io/optimizers/ (https://keras.io/optimizers/))

In [2]:
```
!pip3 install tensorflow keras
```

```
Collecting tensorflow
  Using cached tensorflow-2.16.2-cp310-cp310-win_amd64.whl (2.1 kB)
Requirement already satisfied: keras in c:\users\kaoui\anaconda3\lib
\site-packages (3.4.1)
Collecting tensorflow-intel==2.16.2
  Using cached tensorflow_intel-2.16.2-cp310-cp310-win_amd64.whl (37
6.9 MB)
Requirement already satisfied: astunparse>=1.6.0 in c:\users\kaoui\an
aconda3\lib\site-packages (from tensorflow-intel==2.16.2->tensorflow)
(1.6.3)
Requirement already satisfied: opt-einsum>=2.3.2 in c:\users\kaoui\an
aconda3\lib\site-packages (from tensorflow-intel==2.16.2->tensorflow)
(3.3.0)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in
c:\users\kaoui\anaconda3\lib\site-packages (from tensorflow-intel==2.
16.2->tensorflow) (0.6.0)
Requirement already satisfied: termcolor>=1.1.0 in c:\users\kaoui\ana
conda3\lib\site-packages (from tensorflow-intel==2.16.2->tensorflow)
(2.4.0)
Requirement already satisfied: tensorboard<2.17,>=2.16 in c:\users\ka
oui\anaconda3\lib\site-packages (from tensorflow-intel==2.16.2->tenso
rflow) (2.16.2)
Requirement already satisfied: requests<3,>=2.21.0 in c:\users\kaoui
\anaconda3\lib\site-packages (from tensorflow-intel==2.16.2->tensorfl
ow) (2.28.1)
Requirement already satisfied: typing-extensions>=3.6.6 in c:\users\k
aoui\anaconda3\lib\site-packages (from tensorflow-intel==2.16.2->tens
orflow) (4.12.2)
Requirement already satisfied: numpy<2.0.0,>=1.23.5 in c:\users\kaoui
\anaconda3\lib\site-packages (from tensorflow-intel==2.16.2->tensorfl
ow) (1.23.5)
Requirement already satisfied: setuptools in c:\users\kaoui\anaconda3
\lib\site-packages (from tensorflow-intel==2.16.2->tensorflow) (65.6.
3)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 i
n c:\users\kaoui\anaconda3\lib\site-packages (from tensorflow-intel==
2.16.2->tensorflow) (0.31.0)
Requirement already satisfied: wrapt>=1.11.0 in c:\users\kaoui\anacon
da3\lib\site-packages (from tensorflow-intel==2.16.2->tensorflow) (1.
14.1)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=
4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in c:\users\kaoui\anacond
a3\lib\site-packages (from tensorflow-intel==2.16.2->tensorflow) (4.2
5.3)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in c:\users\kaoui
\anaconda3\lib\site-packages (from tensorflow-intel==2.16.2->tensorfl
ow) (1.64.1)
Requirement already satisfied: libclang>=13.0.0 in c:\users\kaoui\ana
conda3\lib\site-packages (from tensorflow-intel==2.16.2->tensorflow)
(18.1.1)
Requirement already satisfied: ml-dtypes~=0.3.1 in c:\users\kaoui\ana
conda3\lib\site-packages (from tensorflow-intel==2.16.2->tensorflow)
(0.3.2)
Requirement already satisfied: six>=1.12.0 in c:\users\kaoui\anaconda
3\lib\site-packages (from tensorflow-intel==2.16.2->tensorflow) (1.1
6.0)
Requirement already satisfied: flatbuffers>=23.5.26 in c:\users\kaoui
```

```
\anaconda3\lib\site-packages (from tensorflow-intel==2.16.2->tensorfl
ow) (24.3.25)
Requirement already satisfied: absl-py>=1.0.0 in c:\users\kaoui\anaco
nda3\lib\site-packages (from tensorflow-intel==2.16.2->tensorflow)
(2.1.0)
Requirement already satisfied: google-pasta>=0.1.1 in c:\users\kaoui
\anaconda3\lib\site-packages (from tensorflow-intel==2.16.2->tensorfl
ow) (0.2.0)
Requirement already satisfied: h5py>=3.10.0 in c:\users\kaoui\anacond
a3\lib\site-packages (from tensorflow-intel==2.16.2->tensorflow) (3.1
1.0)
Requirement already satisfied: packaging in c:\users\kaoui\anaconda3
\lib\site-packages (from tensorflow-intel==2.16.2->tensorflow) (22.0)
Requirement already satisfied: namex in c:\users\kaoui\anaconda3\lib
\site-packages (from keras) (0.0.8)
Requirement already satisfied: rich in c:\users\kaoui\anaconda3\lib\s
ite-packages (from keras) (13.7.1)
Requirement already satisfied: optree in c:\users\kaoui\anaconda3\lib
\site-packages (from keras) (0.12.1)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in c:\users\ka
oui\anaconda3\lib\site-packages (from rich->keras) (2.18.0)
Requirement already satisfied: markdown-it-py>=2.2.0 in c:\users\kaou
i\anaconda3\lib\site-packages (from rich->keras) (3.0.0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in c:\users\kaoui\a
naconda3\lib\site-packages (from astunparse>=1.6.0->tensorflow-intel=
=2.16.2->tensorflow) (0.38.4)
Requirement already satisfied: mdurl~=0.1 in c:\users\kaoui\anaconda3
\lib\site-packages (from markdown-it-py>=2.2.0->rich->keras) (0.1.2)
Requirement already satisfied: charset-normalizer<3,>=2 in c:\users\k
aoui\anaconda3\lib\site-packages (from requests<3,>=2.21.0->tensorflo
w-intel==2.16.2->tensorflow) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in c:\users\kaoui\anacond
a3\lib\site-packages (from requests<3,>=2.21.0->tensorflow-intel==2.1
6.2->tensorflow) (3.4)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\kaoui\a
naconda3\lib\site-packages (from requests<3,>=2.21.0->tensorflow-inte
l==2.16.2->tensorflow) (2024.6.2)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\users\kaou
i\anaconda3\lib\site-packages (from requests<3,>=2.21.0->tensorflow-i
ntel==2.16.2->tensorflow) (1.26.14)
Requirement already satisfied: markdown>=2.6.8 in c:\users\kaoui\anac
onda3\lib\site-packages (from tensorboard<2.17,>=2.16->tensorflow-int
el==2.16.2->tensorflow) (3.4.1)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0
in c:\users\kaoui\anaconda3\lib\site-packages (from tensorboard<2.17,
>=2.16->tensorflow-intel==2.16.2->tensorflow) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in c:\users\kaoui\anac
onda3\lib\site-packages (from tensorboard<2.17,>=2.16->tensorflow-int
el==2.16.2->tensorflow) (2.2.2)
Requirement already satisfied: MarkupSafe>=2.1.1 in c:\users\kaoui\an
aconda3\lib\site-packages (from werkzeug>=1.0.1->tensorboard<2.17,>=
2.16->tensorflow-intel==2.16.2->tensorflow) (2.1.1)
Installing collected packages: tensorflow-intel, tensorflow
Successfully installed tensorflow-2.16.2 tensorflow-intel-2.16.2
```

In [15]:
```
!pip3 install tabulate
```

```
Requirement already satisfied: tabulate in c:\users\kaoui\anaconda3\l
ib\site-packages (0.8.10)
```

In [16]:
```
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import SGD  #Stochastic Gradient Descent
from tabulate import tabulate

import numpy as np

# fix random seed for reproducibility
np.random.seed(7)

import matplotlib.pyplot as plt
%matplotlib inline
```

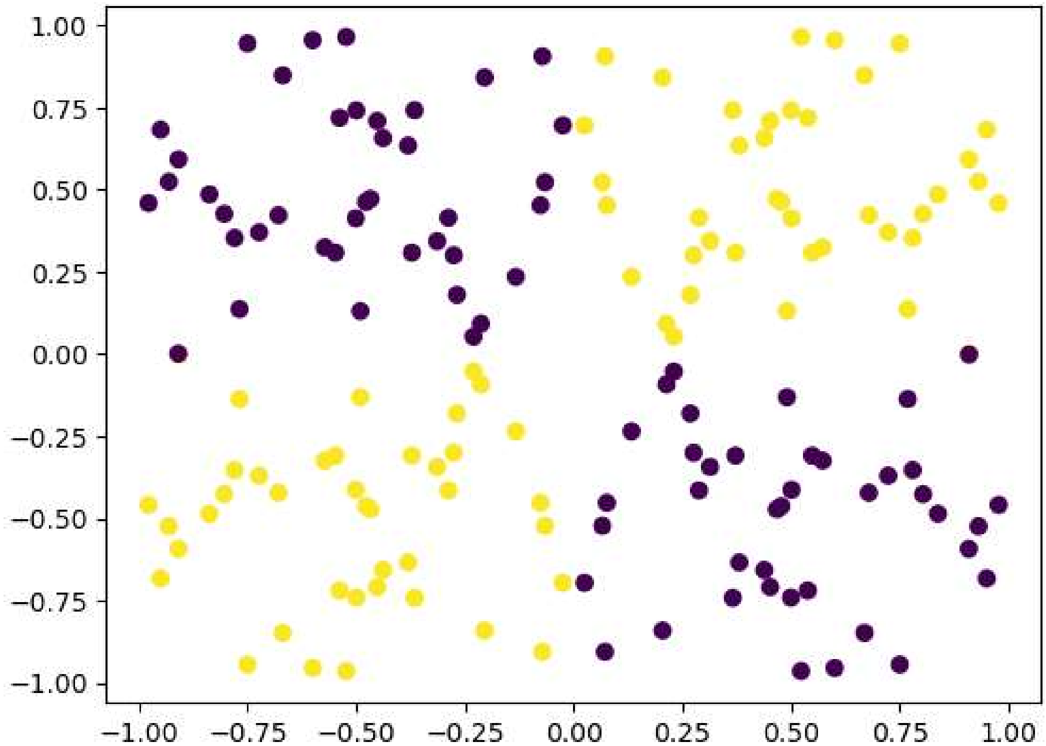
In [4]:
```
n = 40
xx = np.random.random((n,1))
yy = np.random.random((n,1))
```

In [5]:
```
X = np.array([np.array([xx,-xx,-xx,xx]),np.array([yy,-yy,yy,-yy])]).re
y = np.array([np.ones([2*n]),np.zeros([2*n])]).reshape(4*n)
```

In [6]:

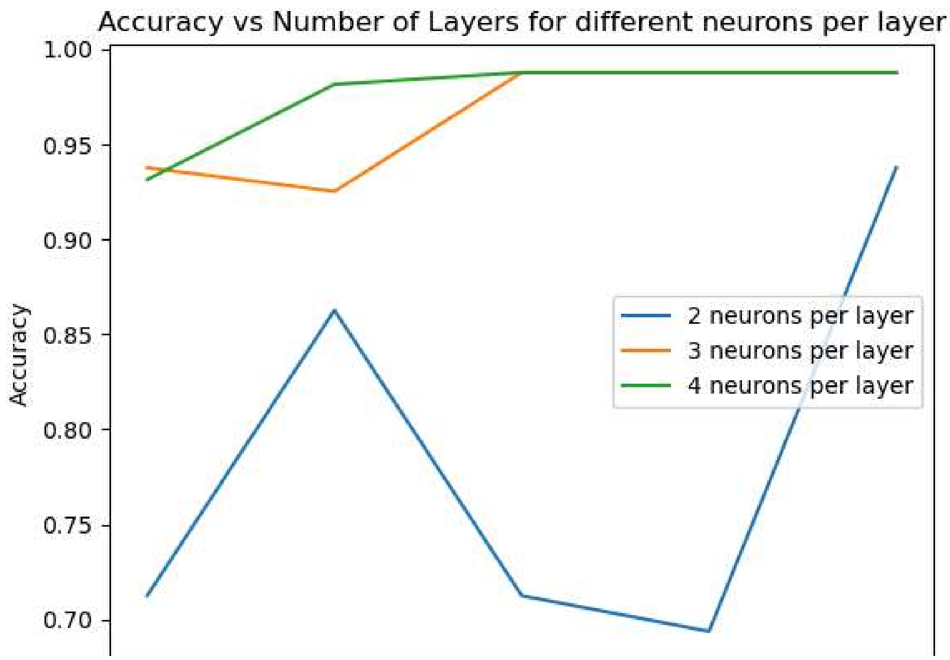
```
plt.scatter(*zip(*X), c=y)
```

Out[6]: <matplotlib.collections.PathCollection at 0x182ac63f940>



In [9]:

```
# Defining function to test different configurations
def build_and_train_model(num_layers, num_neurons, epochs=400):
    model = Sequential()
    model.add(Dense(num_neurons, input_dim=2, activation='tanh'))
    for _ in range(num_layers - 1):
        model.add(Dense(num_neurons, activation='tanh'))
    model.add(Dense(1, activation='sigmoid'))

    model.compile(loss='binary_crossentropy', optimizer=SGD(learning_r
    model.fit(X, y, batch_size=2, epochs=epochs, verbose=0)

    scores = model.evaluate(X, y, verbose=0)
    return scores[1]  # Return accuracy

# Define the number of layers and neurons per layer
num_layers_list = [1, 2, 3, 4, 5]
neurons_per_layer = [2, 3, 4]
epochs = 400

# Store results for plotting
results = {}

for neurons in neurons_per_layer:
    scores = []
    for num_layer in num_layers_list:
        score = build_and_train_model(num_layer, neurons, epochs)
        scores.append(score)
    results[neurons] = scores

# Plot the results
for neurons in neurons_per_layer:
    plt.plot(num_layers_list, results[neurons], label=f'{neurons} neur

plt.xlabel('Number of Layers')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Accuracy vs Number of Layers for different neurons per laye
plt.show()
```



In [17]:

```
# Generating an accuracy table
headers = ['Number of Layers'] + [f'{neurons} Neurons' for neurons in
table_data = []

for i, num_layer in enumerate(num_layers_list):
    row = [num_layer] + [results[neurons][i] for neurons in neurons_pe
    table_data.append(row)

print("Accuracy Rates:")
print(tabulate(table_data, headers=headers))
```

```
Accuracy Rates:
  Number of Layers    2 Neurons    3 Neurons    4 Neurons
------------------  -----------  -----------  -----------
                 1      0.7125        0.9375      0.93125
                 2      0.8625        0.925       0.98125
                 3      0.7125        0.9875      0.9875
                 4      0.69375       0.9875      0.9875
                 5      0.9375        0.9875      0.9875
```

Interpretation: For models with 2 neurons per layer, the optimal number of layers is 5, reaching an accuracy rate of approximately 95%. However, increasing the number of neurons per layer drastically increased the accuracy rate, even for a smaller amount of layers. With that said, the accuracy rate for models containing 3 or more hidden layers with 3 or 4 neurons at each layer could not surpass 98.75%. With that in mind, the most optimal model that minimizes unnecessary complexity and maximizes accuracy is a model with 3 hidden layers with 3 neurons at each layer.

In [18]:

```
from keras.layers import Activation

# Define activation functions to compare
activation_functions = ['tanh', 'sigmoid', 'softplus', 'relu', 'elu',

# Function to build and train model with specified activation function
def build_and_train_model_with_activation(activation, epochs=400):
    model = Sequential()
    model.add(Dense(3, input_dim=2))
    model.add(Activation(activation))  # Add specified activation func
    model.add(Dense(3))
    model.add(Activation(activation))
    model.add(Dense(1, activation='sigmoid'))  # Output layer with sig

    model.compile(loss='binary_crossentropy', optimizer='sgd', metrics
    model.fit(X, y, batch_size=2, epochs=epochs, verbose=0)

    loss, accuracy = model.evaluate(X, y, verbose=0)
    return loss, accuracy

# Storing results for each activation function
activation_results = {}

for activation in activation_functions:
    loss, accuracy = build_and_train_model_with_activation(activation)
    activation_results[activation] = {'loss': loss, 'accuracy': accura

# Print results
print("Activation Function Comparison:")
print("{:<10} {:<10} {:<10}".format('Activation', 'Loss', 'Accuracy'))
print("--------------------------------")
for activation, result in activation_results.items():
    print("{:<10} {:<10.4f} {:<10.4f}".format(activation, result['loss
```

```
Activation Function Comparison:
Activation Loss       Accuracy
----------------------------------
tanh       0.0380     0.9875
sigmoid    0.6922     0.5125
softplus   0.0886     0.9563
relu       0.4902     0.7437
elu        0.0364     0.9875
selu       0.0348     0.9875
swish      0.0956     0.9563
```

Interpretation: In terms of best performers, tanh, elu, and selu activation functions achieved the highest accuracy (98.75%) with relatively low loss (<0.05). The sigmoid activation function performed horribly, achieving an accuracy rate of 51.25% with a very high loss score of 0.69, indicating that it is not suitable for this use case. Softplus and relu activation functions performed decently compared to sigmoid, however they did not achieve as high accuracy scores and had higher loss scores.

Part 2 - BYOD (Bring your own Dataset)

Using your own dataset, experiment and find the best Neural Network configuration. You may use any resource to improve results, just reference it.

While you may use any dataset, I'd prefer you didn't use the diabetes dataset used in the lesson.

[https://stackoverflow.com/questions/34673164/how-to-train-and-tune-an-artificial-multilayer-perceptron-neural-network-using-k (https://stackoverflow.com/questions/34673164/how-to-train-and-tune-an-artificial-multilayer-perceptron-neural-network-using-k)](https://stackoverflow.com/questions/34673164/how-to-train-and-tune-an-artificial-multilayer-perceptron-neural-network-using-k)

[https://keras.io/ (https://keras.io/)](https://keras.io/)

In [24]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping

# Reading data
cancer = pd.read_csv('breastcancer.csv')

# Converting 'diagnosis' into binary variable
cancer['diagnosis'] = cancer['diagnosis'].apply(lambda x: 1 if x == 'M

# Dropping any rows with missing values
cancer.dropna(inplace=True)

# Previewing data
cancer.head()
```

Out[24]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smo
0	842302	1	17.99	10.38	122.80	1001.0	
1	842517	1	20.57	17.77	132.90	1326.0	
2	84300903	1	19.69	21.25	130.00	1203.0	
3	84348301	1	11.42	20.38	77.58	386.1	
4	84358402	1	20.29	14.34	135.10	1297.0	

5 rows × 32 columns

In [25]:

```
# Splitting into X and y
X = cancer.drop(['id', 'diagnosis'], axis=1)  # Features
y = cancer['diagnosis']  # Target variable

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.

# Print shapes to verify data split
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (455, 30)
X_test shape: (114, 30)
y_train shape: (455,)
y_test shape: (114,)
```

In [28]:

```
# Defining new function to test different configurations for breast ca

def build_and_train_model(layers, neurons, activation, optimizer, epoc
    model = Sequential()
    model.add(Dense(neurons, input_dim=X_train.shape[1], activation=ac
    for _ in range(layers - 1):
        model.add(Dense(neurons, activation=activation))
    model.add(Dense(1, activation='sigmoid'))

    model.compile(loss='binary_crossentropy', optimizer=optimizer, met

    # Training the model
    history = model.fit(X_train, y_train, epochs=epochs, batch_size=32

    # Evaluating the model on test data
    loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
    return loss, accuracy, history
```

In [34]:

```
# Define configurations to test
configurations = [
    {'layers': 2, 'neurons': 3, 'activation': 'relu', 'optimizer': 'ad
    {'layers': 3, 'neurons': 4, 'activation': 'tanh', 'optimizer': 'ad
    {'layers': 4, 'neurons': 5, 'activation': 'sigmoid', 'optimizer':
    {'layers': 5, 'neurons': 6, 'activation': 'tanh', 'optimizer': 'ad
    {'layers': 5, 'neurons': 6, 'activation': 'relu', 'optimizer': 'ad
    {'layers': 6, 'neurons': 7, 'activation': 'relu', 'optimizer': 'ad
    {'layers': 5, 'neurons': 15, 'activation': 'relu', 'optimizer': 'a
    {'layers': 6, 'neurons': 7, 'activation': 'softplus', 'optimizer':
    {'layers': 6, 'neurons': 7, 'activation': 'elu', 'optimizer': 'ada
    {'layers': 6, 'neurons': 7, 'activation': 'swish', 'optimizer': 'a
    {'layers': 6, 'neurons': 7, 'activation': 'selu', 'optimizer': 'ad

]

results = []

# Iterate over configurations
for config in configurations:
    layers = config['layers']
    neurons = config['neurons']
    activation = config['activation']
    optimizer = config['optimizer']

    # Build and train model
    loss, accuracy, history = build_and_train_model(layers, neurons, a

    # Store results
    results.append({'layers': layers, 'neurons': neurons, 'activation'
                    'loss': loss, 'accuracy': accuracy, 'history': his

# Print results
print("\nResults:")
print("{:<10} {:<10} {:<10} {:<10} {:<10} {:<10}".format('Layers', 'Ne
print("------------------------------------------------------------")
for result in results:
    print("{:<10} {:<10} {:<10} {:<10} {:<10.4f} {:<10.4f}".format(res
                                                                   res
                                                                   res
```

```
Results:
Layers     Neurons    Activation Optimizer  Loss       Accuracy
------------------------------------------------------------
2          3          relu       adam       0.3750     0.9123
3          4          tanh       adam       0.2226     0.9386
4          5          sigmoid    adam       0.6627     0.6228
5          6          tanh       adam       0.2469     0.9386
5          6          relu       adam       0.5097     0.8158
6          7          relu       adam       0.6630     0.6228
5          15         relu       adam       0.1565     0.9386
6          7          softplus   adam       0.2386     0.9474
6          7          elu        adam       0.1145     0.9561
6          7          swish      adam       0.1972     0.9561
6          7          selu       adam       0.1363     0.9561
```

Interpretation: In testing several differnt models, the best configuration for the breast cancer dataset was a 6 layer model with 7 neurons at each later using an elu activation. This achieved an accuracy score of 95.61% and a relatively low loss score of 0.11. This indicates that the elu activation function and deeper architecture with more neurons per layer were effective for this particular dataset, showcasing its suitability for accurate breast cancer diagnosis prediction.

Data Source: [https://www.kaggle.com/datasets/yasserh/breast-cancer-dataset (https://www.kaggle.com/datasets/yasserh/breast-cancer-dataset)](https://www.kaggle.com/datasets/yasserh/breast-cancer-dataset) Code Help: [https://towardsdatascience.com/how-to-rapidly-test-dozens-of-deep-learning-models-in-python-cb839b518531 (https://towardsdatascience.com/how-to-rapidly-test-dozens-of-deep-learning-models-in-python-cb839b518531)](https://towardsdatascience.com/how-to-rapidly-test-dozens-of-deep-learning-models-in-python-cb839b518531) & ChatGPT was consulted to debug