Building a Basic Neural Network for Image Classification

Artificial Neural Networks (ANNs) are a type of machine learning algorithm that is modeled after the structure and function of the human brain. ANNs are used to learn patterns in data, classify input data, and make predictions based on the learned patterns.

The basic building block of an ANN is a neuron, which receives input signals from other neurons or external sources, processes the inputs, and produces an output signal. The output signal of a neuron can be used as an input to other neurons, forming a network of interconnected neurons. ANNs typically consist of multiple layers of neurons, with each layer performing a different type of computation.

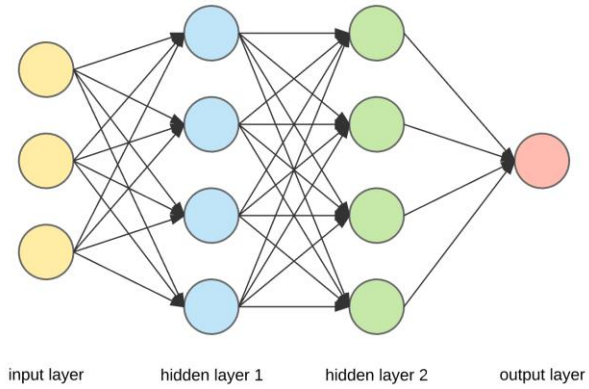
A neuron in an artificial neural network, will perform two operations inside it

- Sum of all weights
- Activation function

So a basic Artificial neural network will be in a form of,

- Input layer – To get the data from the user or a client or a server to analyze and give the result.
- Hidden layers – This layer can be in any number and these layers will analyze the inputs with passing through them with different biases, weights, and activation functions to provide an output

- Output Layer – This is where we can get the result from a neural network.



Some of the features that determine the quality of our neural network are:

1. Layers
2. Activation function
3. Loss function
4. Optimizer

Layers

Layers in a neural network are very important as we saw earlier an artificial neural network consists of 3 layers an input layer, hidden layer, output layer. The input layer consists of the features and values that need to be analyzed inside a neural network. Basically, this is a layer that will read our input features onto an Artificial neural network.

A hidden layer is a layer where all the magic happens when all the input neurons pass the features to the hidden layer with a weight and a bias each and every neuron inside the hidden layer will sum up all the weighted features from all the input layers and apply an activation function to keep the

values between 0 and 1 for easier learning. Here we need to choose the number of neurons in each layer manually and it must be the best value for the network.

There are many types of layers in TensorFlow but the one that we will use is Dense. This is a fully connected layer in which each and every feature input will be connected somehow with the result.

Activation function

Activation functions are simply mathematical methods that bring all the values inside a range of 0 to 1 so that it will be very easier for the machine to learn the data in its process of analyzing the data. Activation functions used by us are sigmoid, ReLu, tanh.

Loss Function

Loss functions in the neural network will calculate the difference between the predicted output and the actual result and greatly help the optimizers in the neural nets to update the weights on its backpropagation.

The loss function we use is Sparse Categorical Crossentropy

Optimizer

Optimizer is the function that helps the neural network to change the weights on the backpropagation so that the difference between the actual and predicted result will decrease at a gradual pace and obtain that point where the loss is very minimum and the model is able to predict more accurate results.

The optimizer we use is adam.

Building the classifier

Importing Libraries

```
# Importing the libraries
import pandas as pd
import numpy as np
from tensorflow import keras
from tensorflow import math
from tensorflow.keras import layers
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from keras.datasets import mnist
```

Loading data

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
X = np.concatenate([x_train, x_test])
y = np.concatenate([y_train, y_test])
X=X/255
```

Splitting data

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.33, random_state=42, shuffle=True)
```

Flattening data

```
X_train_flat=X_train.reshape(len(X_train),28*28)
X_test_flat=X_test.reshape(len(X_test),28*28)
```

We build 15 distinct artificial neural network classifiers by varying one or more paramours from the following list:

- Number of hidden layers – 2 or 3
- Total number of neurons in the hidden layer is 100 or 150
- Activation function is from any of the following functions: tanh, sigmoid, ReLu

Model 1 (2 hidden layers, 100 neurons, activation functions(relu,relu,sigmoid))

```
# Defining the model
```

```
model1 = keras.Sequential([
    keras.layers.Dense(50, input_shape=(28*28,), activation='relu'),
    keras.layers.Dense(50, activation='relu'),
    keras.layers.Dense(10, activation='sigmoid')
])

# Compiling the model
model1.compile(optimizer='adam',
               loss=keras.losses.SparseCategoricalCrossentropy(),
               metrics=['accuracy'])

# fitting the model
model1.fit(X_train_flat, y_train, epochs=10)
```

accuracy=0.9867

Model 2 (2 hidden layers, 150 neurons, activation functions(relu,relu,sigmoid))

```
# Defining the model
model2 = keras.Sequential([
    keras.layers.Dense(100, input_shape=(28*28,), activation='relu'),
    keras.layers.Dense(50, activation='relu'),
    keras.layers.Dense(10, activation='sigmoid')
])

# Compiling the model
model2.compile(optimizer='adam',
               loss=keras.losses.SparseCategoricalCrossentropy(),
               metrics=['accuracy'])

# fitting the model
```

```
model2.fit(X_train_flat, y_train, epochs=10)
```

accuracy=0.9932

Model 3(2 hidden layers, 100 neurons, activation functions(sigmoid,sigmoid, sigmoid))

```
# Defining the model
model4 = keras.Sequential([
    keras.layers.Dense(50, input_shape=(28*28,), activation='sigmoid'),
    keras.layers.Dense(50, activation='sigmoid'),
    keras.layers.Dense(10, activation='sigmoid')
])

# Compiling the model
model4.compile(optimizer='adam',
               loss=keras.losses.SparseCategoricalCrossentropy(),
               metrics=['accuracy'])

# fitting the model
model4.fit(X_train_flat, y_train, epochs=10)
```

accuracy=0.9954

Model 4(2 hidden layers, 150 neurons, activation functions(tanh,relu,sigmoid))

```
# Defining the model
model3 = keras.Sequential([
    keras.layers.Dense(120, input_shape=(28*28,), activation='tanh'),
    keras.layers.Dense(30, activation='relu'),
    keras.layers.Dense(10, activation='sigmoid')
])
```

```
# Compiling the model
model3.compile(optimizer='adam',
               loss=keras.losses.SparseCategoricalCrossentropy(),
               metrics=['accuracy'])

# fitting the model
model3.fit(X_train_flat, y_train, epochs=10)
```

accuracy=0.99831

Model 5(2 hidden layers, 100 neurons, activation functions(tanh,tanh,sigmoid))

```
# Defining the model
model5 = keras.Sequential([
    keras.layers.Dense(50, input_shape=(28*28,), activation='tanh'),
    keras.layers.Dense(50, activation='tanh'),
    keras.layers.Dense(10, activation='sigmoid')
])

# Compiling the model
model5.compile(optimizer='adam',
               loss=keras.losses.SparseCategoricalCrossentropy(),
               metrics=['accuracy'])

# fitting the model
model5.fit(X_train_flat, y_train, epochs=10)
```

accuracy=0.9907

Model 6(3 hidden layers, 150 neurons, activation functions(relu,relu,relu,sigmoid))

```
# Defining the model
model6 = keras.Sequential([
    keras.layers.Dense(50, input_shape=(28*28,), activation='relu'),
    keras.layers.Dense(50, activation='relu'),
    keras.layers.Dense(50, activation='relu'),
    keras.layers.Dense(10, activation='sigmoid')
])

# Compiling the model
model6.compile(optimizer='adam',
               loss=keras.losses.SparseCategoricalCrossentropy(),
               metrics=['accuracy'])

# fitting the model
model6.fit(X_train_flat, y_train, epochs=10)
```

accuracy=0.9879

Model 7(3 hidden layers, 150 neurons, activation functions(tanh,relu,relu,sigmoid))

```
# Defining the model
model7 = keras.Sequential([
    keras.layers.Dense(80, input_shape=(28*28,), activation='tanh'),
    keras.layers.Dense(40, activation='relu'),
    keras.layers.Dense(30, activation='relu'),
    keras.layers.Dense(10, activation='sigmoid')
])
```

```
# Compiling the model
model7.compile(optimizer='adam',
               loss=keras.losses.SparseCategoricalCrossentropy(),
               metrics=['accuracy'])

# fitting the model
model7.fit(X_train_flat, y_train, epochs=10)
```

accuracy=0.9919

Model 8(3 hidden layers, 150 neurons, activation functions(relu,tanh,tanh,sigmoid))

```
# Defining the model
model8 = keras.Sequential([
    keras.layers.Dense(90, input_shape=(28*28,), activation='relu'),
    keras.layers.Dense(40, activation='tanh'),
    keras.layers.Dense(20, activation='tanh'),
    keras.layers.Dense(10, activation='sigmoid')
])

# Compiling the model
model8.compile(optimizer='adam',
               loss=keras.losses.SparseCategoricalCrossentropy(),
               metrics=['accuracy'])

# fitting the model
model8.fit(X_train_flat, y_train, epochs=10)
```

accuracy=0.9921

Model 9(3 hidden layers, 150 neurons, activation functions(tanh,tanh,tanh,,sigmoid))

```
# Defining the model
model9 = keras.Sequential([
    keras.layers.Dense(105, input_shape=(28*28,), activation='tanh'),
    keras.layers.Dense(30, activation='tanh'),
    keras.layers.Dense(15, activation='tanh'),
    keras.layers.Dense(10, activation='sigmoid')
])

# Compiling the model
model9.compile(optimizer='adam',
               loss=keras.losses.SparseCategoricalCrossentropy(),
               metrics=['accuracy'])

# fitting the model
model9.fit(X_train_flat, y_train, epochs=10)
```

accuracy=0.9918

Model 10(3 hidden layers, 150 neurons, activation functions(tanh,,tanh,sigmoid, sigmoid))

```
# Defining the model
model8 = keras.Sequential([
    keras.layers.Dense(90, input_shape=(28*28,), activation='tanh'),
    keras.layers.Dense(40, activation='tanh'),
    keras.layers.Dense(20, activation='sigmoid'),
    keras.layers.Dense(10, activation='sigmoid')
])
```

```
# Compiling the model
model8.compile(optimizer='adam',
               loss=keras.losses.SparseCategoricalCrossentropy(),
               metrics=['accuracy'])

# fitting the model
model8.fit(X_train_flat, y_train, epochs=10)
```

accuracy=0.9922

Model 11(3 hidden layers, 100 neurons, activation functions(relu,tanh,tanh,sigmoid))

```
# Defining the model
model11 = keras.Sequential([
    keras.layers.Dense(50, input_shape=(28*28,), activation='relu'),
    keras.layers.Dense(30, activation='tanh'),
    keras.layers.Dense(20, activation='tanh'),
    keras.layers.Dense(10, activation='sigmoid')
])

# Compiling the model
model11.compile(optimizer='adam',
               loss=keras.losses.SparseCategoricalCrossentropy(),
               metrics=['accuracy'])

# fitting the model
model11.fit(X_train_flat, y_train, epochs=10)
```

accuracy=0.9888

Model 12(3 hidden layers, 150 neurons, activation functions(sigmoid,sigmoid,tanh, ,sigmoid))

```
# Defining the model
model12 = keras.Sequential([
    keras.layers.Dense(90, input_shape=(28*28,), activation='sigmoid'),
    keras.layers.Dense(40, activation='sigmoid'),
    keras.layers.Dense(20, activation='tanh'),
    keras.layers.Dense(10, activation='sigmoid')
])

# Compiling the model
model12.compile(optimizer='adam',
              loss=keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])

# fitting the model
model12.fit(X_train_flat, y_train, epochs=10)
```

accuracy=0.9904

Model 13(3 hidden layers, 100 neurons, activation functions(sigmoid,relu,sigmoid, sigmoid))

```
# Defining the model
model11 = keras.Sequential([
    keras.layers.Dense(50, input_shape=(28*28,), activation='sigmoid'),
    keras.layers.Dense(30, activation='relu'),
    keras.layers.Dense(20, activation='sigmoid'),
    keras.layers.Dense(10, activation='sigmoid')
])
```

```
# Compiling the model
model11.compile(optimizer='adam',
                loss=keras.losses.SparseCategoricalCrossentropy(),
                metrics=['accuracy'])

# fitting the model
model11.fit(X_train_flat, y_train, epochs=10)
```

accuracy=0.9831

Model 14(2 hidden layers, 100 neurons, activation functions(sigmoid,relu,sigmoid))

```
# Defining the model
model11 = keras.Sequential([
    keras.layers.Dense(80, input_shape=(28*28,), activation='sigmoid'),
    keras.layers.Dense(20, activation='relu'),
    keras.layers.Dense(10, activation='sigmoid')
])

# Compiling the model
model11.compile(optimizer='adam',
                loss=keras.losses.SparseCategoricalCrossentropy(),
                metrics=['accuracy'])

# fitting the model
model11.fit(X_train_flat, y_train, epochs=10)
```

accuracy=0.9883

Model 15(2 hidden layers, 100 neurons, activation functions(relu,tanh,sigmoid))

```
# Defining the model
```

```
model11 = keras.Sequential([
    keras.layers.Dense(75, input_shape=(28*28,), activation='relu'),
    keras.layers.Dense(25, activation='tanh'),
    keras.layers.Dense(10, activation='sigmoid')
])

# Compiling the model
model11.compile(optimizer='adam',
                loss=keras.losses.SparseCategoricalCrossentropy(),
                metrics=['accuracy'])

# fitting the model
model11.fit(X_train_flat, y_train, epochs=10)
```

accuracy=0.9926

Confusion Matrices:

```
722/722 [==============================] - 1s 1ms/step
[[2269    0    1    0    2    0    1    0    2    0]
 [   0 2617    0    0    0    0    0    0    0    0]
 [   0   52 2256    2    0    0    0    0    1    0]
 [   0   13    6 2332    0    1    0    2    1    2]
 [   0    7    2    0 2224    0    0    1    0    0]
 [   3    0    0  131    0 1918    0    0    0    5]
 [   8    5    0    0    1   26 2280    1    1    0]
 [   0    7    6    4    0    0    0 2292    1    0]
 [   0   16   10    8    2    0    0    1 2200    4]
 [   0    3    0    2   33    0    0   21    3 2314]]
722/722 [==============================] - 1s 2ms/step
[[2272    0    0    0    0    0    0    0    1    2]
```

```
 [   0 2614    3    0    0    0    0    0    0    0]
 [   0    1 2309    0    0    0    0    0    1    0]
 [   1    0    2 2346    0    0    0    4    2    2]
 [   0    0    5    0 2227    0    0    1    0    1]
 [   1    0    1   49    1 2001    3    0    0    1]
 [   6    0    1    0    1    1 2312    0    1    0]
 [   0    1    1    0    1    1    0 2305    0    1]
 [   0    2    1    0    4    1    0    1 2231    1]
 [   0    0    0    0   10    0    0    2    1 2363]]
722/722 [==============================] - 2s 2ms/step
[[2270    0    0    0    1    2    0    0    2    0]
 [   0 2617    0    0    0    0    0    0    0    0]
 [   0    0 2307    3    0    0    0    0    1    0]
 [   0    0    0 2356    0    1    0    0    0    0]
 [   0    0    0    0 2231    0    0    0    0    3]
 [   0    0    0   15    0 2041    0    0    1    0]
 [   0    1    0    0    0    1 2319    1    0    0]
 [   0    0    0    2    0    0    0 2307    0    1]
 [   0    0    0    1    0    1    0    1 2238    0]
 [   0    1    0    1   42    2    0   11    0 2319]]
722/722 [==============================] - 1s 2ms/step
[[2267    0    2    0    1    1    2    0    2    0]
 [   1 2612    2    1    1    0    0    0    0    0]
 [   0    1 2306    0    0    1    0    0    2    1]
 [   1    0    1 2347    0    3    0    1    1    3]
 [   0    0    1    0 2226    0    0    1    0    6]
 [   1    0    0    0    1 2052    0    1    1    1]
 [   0    0    0    0    0    1 2319    0    2    0]
 [   0    0    1    0    1    0    0 2304    1    3]
```

```
 [   0    0    0    0    1    0    0    1 2239    0]
 [   0    0    0    2    2    1    0    2    0 2369]]
722/722 [==============================] - 1s 2ms/step
[[2256    0    3    1    3    0    4    0    5    3]
 [   0 2604    7    1    0    0    0    1    4    0]
 [   0    0 2304    1    2    0    1    0    2    1]
 [   2    1    7 2340    0    1    0    2    2    2]
 [   0    1    1    0 2220    0    5    3    2    2]
 [   0    0    0    3    1 2050    1    0    0    2]
 [   0    1    1    0    0    5 2312    0    3    0]
 [   0    0   15    3    1    0    0 2290    0    1]
 [   0    0    9    3    0    0    1    1 2226    1]
 [   0    1    0    1   11    1    0   11    4 2347]]
722/722 [==============================] - 1s 2ms/step
[[2269    0    1    0    0    1    2    0    2    0]
 [   0 2608    2    1    0    0    0    3    1    2]
 [   0    0 2307    0    1    0    0    2    0    1]
 [   0    1    7 2344    0    1    0    1    2    1]
 [   1    0    0    0 2226    0    1    1    1    4]
 [   0    0    1  228    0 1824    2    1    0    1]
 [   2    1    0    0    0    5 2311    0    3    0]
 [   0    1    6    1    2    0    0 2300    0    0]
 [   1    1   15    2    0    3    3    1 2211    4]
 [   0    0    1    3    3    5    0   10    2 2352]]
722/722 [==============================] - 1s 1ms/step
[[2270    0    0    0    0    0    4    0    1    0]
 [   0 2614    1    0    1    0    1    0    0    0]
 [   3    1 2299    0    3    2    1    0    1    1]
 [   1    1    1 2333    1    9    0    1    5    5]
```

```
 [   0    0    1    0 2227    0    4    1    0    1]
 [   2    0    0    0    0 2052    1    0    1    1]
 [   2    0    0    0    0    3 2316    0    1    0]
 [   3    1    0    1    1    3    0 2299    0    2]
 [   1    3    1    0    0    2    3    1 2228    2]
 [   0    1    0    0    6    0    1    0    0 2368]]
722/722 [==============================] - 1s 2ms/step
[[2266    0    3    0    2    0    2    0    1    1]
 [   0 2613    0    1    0    1    2    0    0    0]
 [   0    1 2300    7    0    1    0    1    1    0]
 [   0    0    0 2354    0    0    0    1    0    2]
 [   0    0    1    0 2229    0    1    1    0    2]
 [   0    0    0    3    0 2051    1    0    0    2]
 [   0    1    0    0    6    2 2313    0    0    0]
 [   1    1    1    1    1    0    0 2302    1    2]
 [   0    1    4    8    1    5    1    1 2218    2]
 [   0    1    0    3    1    1    0    1    0 2369]]
722/722 [==============================] - 1s 2ms/step
[[2266    0    2    1    2    1    1    1    1    0]
 [   0 2615    1    0    0    0    0    0    1    0]
 [   2    2 2299    3    0    1    0    1    2    1]
 [   1    1    1 2344    0    1    0    5    2    2]
 [   0    0    0    0 2228    0    1    1    1    3]
 [   0    0    0    2    1 2047    1    0    3    3]
 [   0    2    0    0    3    7 2306    0    4    0]
 [   0    1    2    1    3    0    0 2301    1    1]
 [   1    4    2    5    1    1    0    1 2225    1]
 [   3    2    0    1    4    0    0    8    1 2357]]
722/722 [==============================] - 1s 2ms/step
```

```
[[2268    0    2    0    2    0    1    0    2    0]
 [   0 2608    4    0    1    0    0    2    2    0]
 [   1    0 2308    0    0    0    0    1    1    0]
 [   1    0    4 2347    0    2    0    0    1    2]
 [   0    0    2    0 2227    1    1    2    0    1]
 [   0    1    0    0    1 2052    1    0    0    2]
 [   2    0    0    0    0    3 2316    0    1    0]
 [   0    0    0    0    1    0    0 2308    1    0]
 [   1    1    5    4    2    2    2    1 2222    1]
 [   2    0    1    0    7    2    0   15    2 2347]]
722/722 [==============================] - 1s 2ms/step
[[2271    0    0    0    1    1    0    0    2    0]
 [   0 2615    1    0    0    0    0    1    0    0]
 [   1    1 2302    0    2    0    0    0    4    1]
 [   1    0    3 2346    0    0    0    1    3    3]
 [   0    0    1    0 2217    0    2    1    0   13]
 [   0    0    0    1    1 2052    0    0    1    2]
 [   3    1    0    1    1    5 2310    0    1    0]
 [   0    0    1    2    1    0    0 2299    2    5]
 [   0    0    1    2    0    1    1    1 2234    1]
 [   0    0    0    0    0    1    0    2    2 2371]]
722/722 [==============================] - 1s 2ms/step
[[2272    0    1    0    0    0    1    0    0    1]
 [   0 2612    1    0    1    0    0    0    3    0]
 [   1    0 2291   11    2    1    0    2    3    0]
 [   1    0    1 2353    0    2    0    0    0    0]
 [   0    0    2    0 2225    0    1    1    0    5]
 [   0    0    0    1    0 2053    0    0    0    3]
 [   1    0    0    0    1   13 2306    0    1    0]
```

```
 [   0    0    0    3    1    1    0 2302    1    2]
 [   0    0    0    2    0    2    0    0 2234    3]
 [   1    0    0    0    2    1    0    1    0 2371]]
722/722 [==============================] - 1s 2ms/step
[[2267    0    1    1    0    0    4    1    1    0]
 [   0 2610    5    1    1    0    0    0    0    0]
 [   0    2 2302    1    2    0    1    1    2    0]
 [   1    0    5 2344    0    3    0    1    2    1]
 [   1    0    2    1 2223    0    0    1    1    5]
 [   1    0    1    5    1 2037    6    1    2    3]
 [   2    1    0    0    0    2 2316    0    1    0]
 [   1    2    3    3    4    0    0 2291    2    4]
 [   0    2    3    1    0    2    1    0 2230    2]
 [   3    1    0    1    7    1    0    2    3 2358]]
722/722 [==============================] - 1s 2ms/step
[[2267    0    0    0    3    0    3    0    1    1]
 [   0 2608    2    0    3    0    0    0    2    2]
 [   0    0 2304    1    1    0    1    0    3    1]
 [   0    0    9 2326    0    4    0    3    6    9]
 [   1    0    2    0 2221    0    2    0    0    8]
 [   1    0    1    3    3 2030    6    1    2   10]
 [   2    1    0    0    2    1 2314    0    2    0]
 [   1    2    4    1    4    1    0 2280    1   16]
 [   0    2    2    2    1    0    0    0 2232    2]
 [   1    0    0    1    5    0    0    0    1 2368]]
722/722 [==============================] - 1s 2ms/step
[[2268    0    2    0    2    0    3    0    0    0]
 [   0 2614    0    0    0    0    1    0    2    0]
 [   1    0 2297    0    1    1    1    3    6    1]
```

```
[   0    0    1 2349    0    2    0    1    3    1]
[   0    0    1    0 2224    0    3    2    1    3]
[   3    0    0    5    2 2044    1    0    1    1]
[   1    0    1    0    1    3 2316    0    0    0]
[   1    1    1    0    3    0    1 2301    1    1]
[   0    1    0    3    1    1    2    1 2231    1]
[   0    1    0    3    9    1    0    3    5 2354]]
```

Based on the above metrics, we infer that model4 is the best model.

Average accuracy=0.998168820142746

Confusion matrix:

	0	1	2	3	4	5	6	7	8	9
0	2270.0	0.0	0.0	0.0	1.0	2.0	0.0	0.0	2.0	0.0
1	0.0	2617.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	2307.0	3.0	0.0	0.0	0.0	0.0	1.0	0.0
3	0.0	0.0	0.0	2356.0	0.0	1.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	2231.0	0.0	0.0	0.0	0.0	3.0
5	0.0	0.0	0.0	15.0	0.0	2041.0	0.0	0.0	1.0	0.0
6	0.0	1.0	0.0	0.0	0.0	1.0	2319.0	1.0	0.0	0.0
7	0.0	0.0	0.0	2.0	0.0	0.0	0.0	2307.0	0.0	1.0
8	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	2238.0	0.0
9	0.0	1.0	0.0	1.0	42.0	2.0	0.0	11.0	0.0	2319.0

Model15 is not statistically significant from the best classifier as the accuracy and other metrics differ only slightly.

ANNs