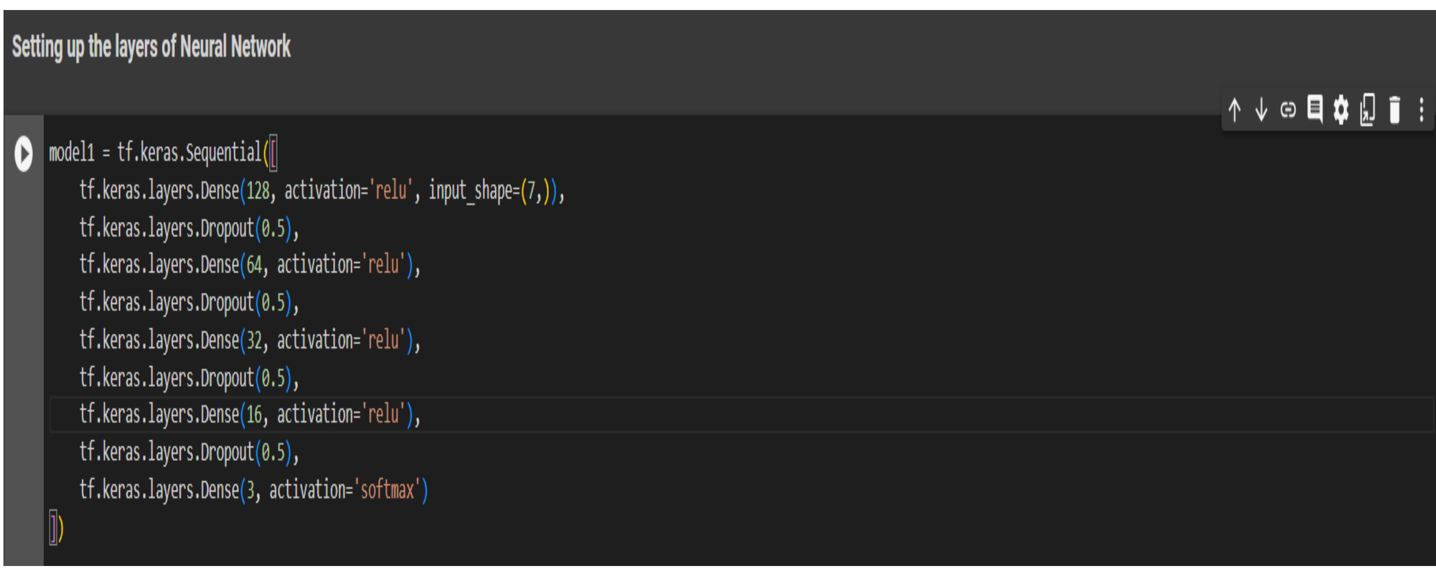
Function 4(Research on fish diseases in aquaculture mainly investigates the individual effects of water quality parameters)

Our machine learning model is intricately designed to provide predictions regarding the probability and severity of fish diseases in aquaculture. By leveraging advanced algorithms, we systematically take into account a comprehensive set of water quality parameters, including temperature, dissolved oxygen levels, salinity, pH, nitrate, and ammonia. Through the simultaneous analysis of these multiple factors, we aim to discern patterns and relationships that contribute to the occurrence and intensity of diseases such as Red Spot, White Spot (Ich), and Fin Rot. This holistic approach enables our team to develop a more nuanced understanding of the interplay between water quality and fish health, allowing for proactive measures to mitigate disease risks and optimize the overall well-being of the aquatic population.

We create two neural networks from scratch.These architectures are meticulously designed to handle feature extraction and disease prediction, respectively, focusing on intricate patterns within water quality parameters. Following this, we fine-tune and optimize these architectures by adjusting hyperparameters, aiming for the best possible results in predicting and mitigating fish diseases in aquaculture.

Model_Architecture_1

```
model1 = tf.keras.Sequential([
    tf.keras.layers.Dense(128, activation='relu', 
input_shape=(7,)),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(3, activation='softmax')
])
```



Input Layer:

Type: Dense layer
Neurons: 128
Activation Function: Rectified Linear Unit (ReLU)
Input Shape: (7,)
Description: This layer serves as the input layer with 128 neurons, each using the Rectified Linear Unit (ReLU) activation function. The input shape is set to (7,), indicating that the model expects input data with seven features.

Dropout Layer 1:

Type: Dropout layer
Rate: 0.5
Description: Dropout is a regularization technique that helps prevent overfitting by randomly setting a fraction of input units to zero during training. In this case, 50% of the neurons from the previous layer are dropped out during each training iteration.

Hidden Layer 1:

Type: Dense layer
Neurons: 64
Activation Function: ReLU
Description: This is the first hidden layer with 64 neurons and ReLU activation. It helps the model learn complex patterns and relationships within the data.

Dropout Layer 2:

Type: Dropout layer
Rate: 0.5
Description: Another dropout layer with a rate of 0.5 is added after the first hidden layer.

Hidden Layer 2:

Type: Dense layer
Neurons: 32
Activation Function: ReLU
Description: This is the second hidden layer with 32 neurons and ReLU activation.

Dropout Layer 3:

Type: Dropout layer
Rate: 0.5
Description: Another dropout layer with a rate of 0.5 is added after the second hidden layer.

Hidden Layer 3:

Type: Dense layer
Neurons: 16
Activation Function: ReLU
Description: This is the third hidden layer with 16 neurons and ReLU activation.

Dropout Layer 4:

Type: Dropout layer
Rate: 0.5
Description: Another dropout layer with a rate of 0.5 is added after the third hidden layer.

Output Layer:

Type: Dense layer
Neurons: 3
Activation Function: Softmax

Description: This is the output layer with 3 neurons and softmax activation. The softmax activation is often used in multi-class classification problems, as it converts the network's output into probability distributions over the three classes.

We trained three models using the specified architecture, fine-tuning hyperparameters for optimal performance, and achieved ultimate results.

Model_1

Setting up the layers of Neural Network

```
model1 = tf.keras.Sequential([
    tf.keras.layers.Dense(128, activation='relu', input_shape=(7,)),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(3, activation='softmax')
])
```

compile the model

```
model1.compile(
    loss=tf.keras.losses.categorical_crossentropy,
    optimizer=tf.keras.optimizers.RMSprop(learning_rate=0.001),
    metrics=['accuracy']
)
```

Train the Model

```
epoch_number = 20
history = model1.fit(x_train, y_train, epochs=epoch_number,batch_size=256)
```

```
Epoch 1/20
1626/1626 [==============================] - 9s 5ms/step - loss: 0.3364 - accuracy: 0.8801
Epoch 2/20
1626/1626 [==============================] - 6s 4ms/step - loss: 0.1385 - accuracy: 0.9619
Epoch 3/20
1626/1626 [==============================] - 7s 5ms/step - loss: 0.0968 - accuracy: 0.9759
Epoch 4/20
1626/1626 [==============================] - 6s 4ms/step - loss: 0.0853 - accuracy: 0.9815
Epoch 5/20
1626/1626 [==============================] - 8s 5ms/step - loss: 0.0787 - accuracy: 0.9836
```

```
Epoch 16/20
1626/1626 [==============================] - 8s 5ms/step - loss: 0.0914 - accuracy: 0.9877
Epoch 17/20
1626/1626 [==============================] - 6s 4ms/step - loss: 0.0798 - accuracy: 0.9878
Epoch 18/20
1626/1626 [==============================] - 8s 5ms/step - loss: 0.0838 - accuracy: 0.9875
Epoch 19/20
1626/1626 [==============================] - 7s 4ms/step - loss: 0.0878 - accuracy: 0.9876
Epoch 20/20
1626/1626 [==============================] - 10s 6ms/step - loss: 0.0805 - accuracy: 0.9878
```

Model Summery and Performance

```
#summary of the neural network model
model1.summary()
```

```
Model: "sequential_16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_62 (Dense)            (None, 128)               1024

 dropout_24 (Dropout)        (None, 128)               0

 dense_63 (Dense)            (None, 64)                8256

 dropout_25 (Dropout)        (None, 64)                0

 dense_64 (Dense)            (None, 32)                2080

 dropout_26 (Dropout)        (None, 32)                0

 dense_65 (Dense)            (None, 16)                528

 dropout_27 (Dropout)        (None, 16)                0

 dense_66 (Dense)            (None, 3)                 51

=================================================================
Total params: 11939 (46.64 KB)
Trainable params: 11939 (46.64 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Evaluate Model

```
model1.evaluate(x_test,y_test)
```

```
5572/5572 [==============================] - 15s 3ms/step - loss: 0.0309 - accuracy: 0.9967
[0.030934499576687813, 0.9966740608215332]
```

Precision

```
from sklearn.preprocessing import LabelBinarizer
label_binarizer = LabelBinarizer()
y_true_one_hot = label_binarizer.fit_transform(y_test)

# Convert the one-hot encoded true labels to class labels
y_true_classes = tf.argmax(y_true_one_hot, axis=1)

# Precision
precision = tf.keras.metrics.Precision()
precision.update_state(y_true_classes, y_pred)
precision.result()
```

```
<tf.Tensor: shape=(), dtype=float32, numpy=0.99794686>
```

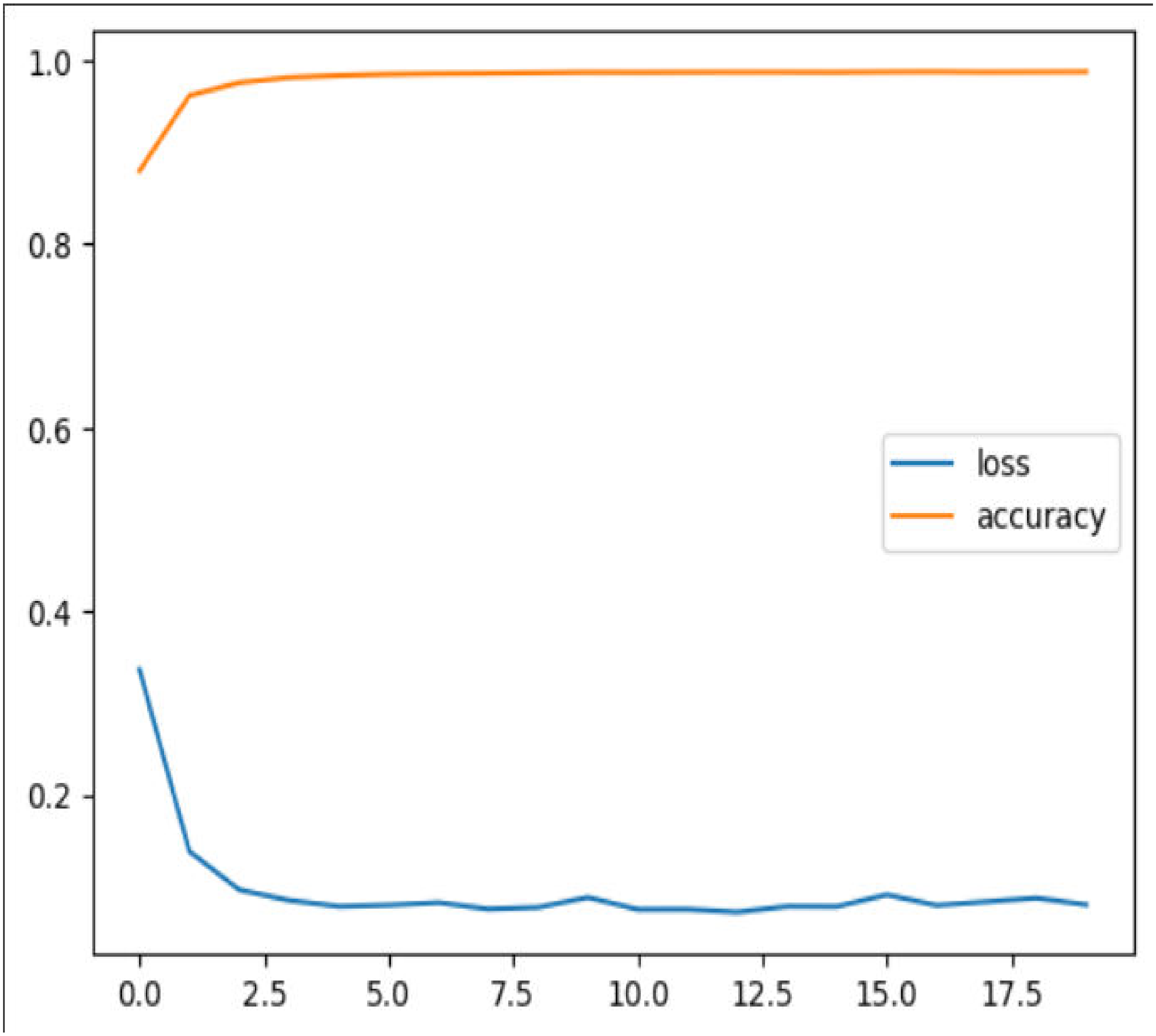
Recall

```
# Recall
recall = tf.keras.metrics.Recall()
recall.update_state(y_true_classes, y_pred)
recall.result()
```

```
<tf.Tensor: shape=(), dtype=float32, numpy=0.9975112>
```

- ✓ **Training_Accuracy -> 98.78%**
- ✓ **Testing_Accuracy -> 99.66%**
- ✓ **Precision -> 99.79%**
- ✓ **Recall -> 99.75%**
- ✓ **No.Epochs -> 20**
- ✓ **Batch_Size -> 256**
- ✓ **Learning_Rate -> 0.001**
- ✓ **Optimizer -> RMSprop**

Training Accuracy and Loss Over Epochs:



Model_2

Setting up the layers of Neural Network

```
model2 = tf.keras.Sequential([
    tf.keras.layers.Dense(128, activation='relu', input_shape=(7,)),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(3, activation='softmax')
])
```

compile the model

```
model2.compile(
    loss=tf.keras.losses.categorical_crossentropy,
    optimizer=tf.keras.optimizers.SGD(learning_rate=0.01),
    metrics=['accuracy']
)
```

Train the Model

```
epoch_number = 15
history = model2.fit(x_train, y_train, epochs=epoch_number,batch_size=256)
```

```
Epoch 1/15
1626/1626 [==============================] - 10s 5ms/step - loss: 0.9250 - accuracy: 0.5469
Epoch 2/15
1626/1626 [==============================] - 7s 5ms/step - loss: 0.6392 - accuracy: 0.7604
Epoch 3/15
1626/1626 [==============================] - 9s 6ms/step - loss: 0.4956 - accuracy: 0.8283
Epoch 4/15
1626/1626 [==============================] - 8s 5ms/step - loss: 0.4073 - accuracy: 0.8664
Epoch 5/15
1626/1626 [==============================] - 8s 5ms/step - loss: 0.3409 - accuracy: 0.8926
```

```
Epoch 10/15
1626/1626 [==============================] - 9s 6ms/step - loss: 0.1927 - accuracy: 0.9420
Epoch 11/15
1626/1626 [==============================] - 7s 4ms/step - loss: 0.1810 - accuracy: 0.9464
Epoch 12/15
1626/1626 [==============================] - 10s 6ms/step - loss: 0.1653 - accuracy: 0.9506
Epoch 13/15
1626/1626 [==============================] - 8s 5ms/step - loss: 0.1562 - accuracy: 0.9538
Epoch 14/15
1626/1626 [==============================] - 10s 6ms/step - loss: 0.1518 - accuracy: 0.9551
Epoch 15/15
1626/1626 [==============================] - 6s 4ms/step - loss: 0.1420 - accuracy: 0.9582
```

Model Summery and Performance

Evaluate Model

```
model2.evaluate(x_test,y_test)
```

```
5572/5572 [==============================] - 15s 3ms/step - loss: 0.0456 - accuracy: 0.9890
[0.04557088017463684, 0.9889733791351318]
```

Model Summary

```
#summary of the neural network model
model2.summary()
```

```
Model: "sequential_17"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_67 (Dense)            (None, 128)               1024

 dropout_28 (Dropout)        (None, 128)               0

 dense_68 (Dense)            (None, 64)                8256

 dropout_29 (Dropout)        (None, 64)                0

 dense_69 (Dense)            (None, 32)                2080

 dropout_30 (Dropout)        (None, 32)                0

 dense_70 (Dense)            (None, 16)                528

 dropout_31 (Dropout)        (None, 16)                0

 dense_71 (Dense)            (None, 3)                 51

=================================================================
Total params: 11939 (46.64 KB)
Trainable params: 11939 (46.64 KB)
Non-trainable params: 0 (0.00 Byte)
```

Precision

```
from sklearn.preprocessing import LabelBinarizer
label_binarizer = LabelBinarizer()
y_true_one_hot = label_binarizer.fit_transform(y_test)

# Convert the one-hot encoded true labels to class labels
y_true_classes = tf.argmax(y_true_one_hot, axis=1)

# Precision
precision = tf.keras.metrics.Precision()
precision.update_state(y_true_classes, y_pred)
precision.result()
```

```
<tf.Tensor: shape=(), dtype=float32, numpy=0.99310035>
```

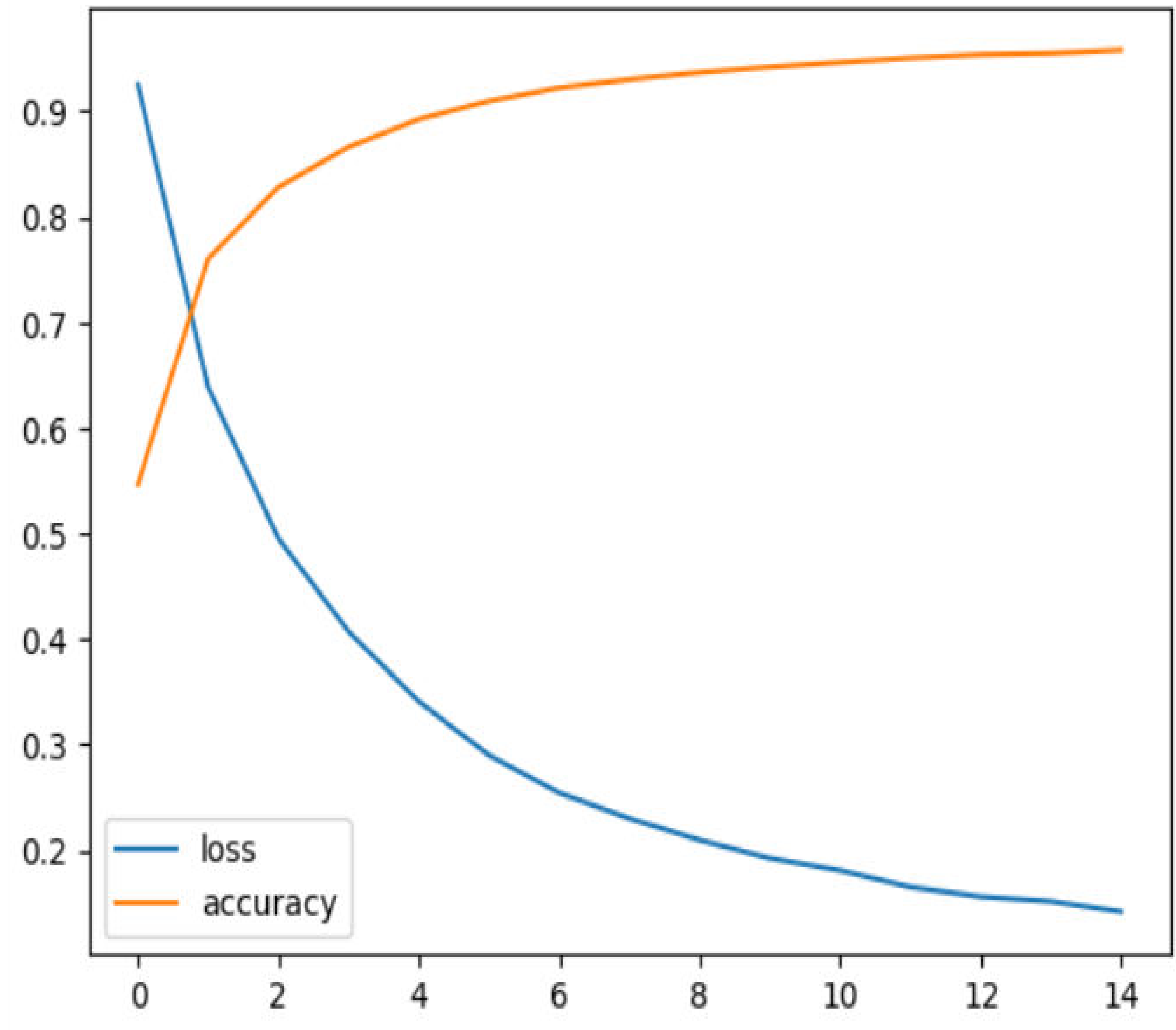
Recall

```
# Recall
recall = tf.keras.metrics.Recall()
recall.update_state(y_true_classes, y_pred)
recall.result()
```

```
<tf.Tensor: shape=(), dtype=float32, numpy=0.9930927>
```

- ✓ **Training_Accuracy -> 95.82%**
- ✓ **Testing_Accuracy -> 98.89%**
- ✓ **Precision -> 99.31%**
- ✓ **Recall -> 99.30%**
- ✓ **No.Epochs -> 15**
- ✓ **Batch_Size -> 256**
- ✓ **Learning_Rate -> 0.01**
- ✓ **Optimizer -> SGD**

Training Accuracy and Loss Over Epochs



Model_3

Setting up the layers of Neural Network

```
[ ] model3 = tf.keras.Sequential([
        tf.keras.layers.Dense(128, activation='relu', input_shape=(7,)),
        tf.keras.layers.Dropout(0.5),
        tf.keras.layers.Dense(64, activation='relu'),
        tf.keras.layers.Dropout(0.5),
        tf.keras.layers.Dense(32, activation='relu'),
        tf.keras.layers.Dropout(0.5),
        tf.keras.layers.Dense(16, activation='relu'),
        tf.keras.layers.Dropout(0.5),
        tf.keras.layers.Dense(3, activation='softmax')
      ])
```

compile the model

+ Code + Text

```
[ ] model3.compile(
        loss=tf.keras.losses.categorical_crossentropy,
        optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
        metrics=['accuracy']
      )
```

Train the Model

```
[ ] epoch_number = 15
    history = model3.fit(x_train, y_train, epochs=epoch_number,batch_size=128)
```

```
Epoch 1/15
3251/3251 [==============================] - 16s 4ms/step - loss: 0.2951 - accuracy: 0.9000
Epoch 2/15
3251/3251 [==============================] - 14s 4ms/step - loss: 0.1161 - accuracy: 0.9690
Epoch 3/15
3251/3251 [==============================] - 14s 4ms/step - loss: 0.0854 - accuracy: 0.9765
Epoch 4/15
3251/3251 [==============================] - 14s 4ms/step - loss: 0.0758 - accuracy: 0.9789
Epoch 5/15
3251/3251 [==============================] - 18s 5ms/step - loss: 0.0663 - accuracy: 0.9805
```

```
Epoch 10/15
3251/3251 [==============================] - 14s 4ms/step - loss: 0.0535 - accuracy: 0.9838
Epoch 11/15
3251/3251 [==============================] - 17s 5ms/step - loss: 0.0534 - accuracy: 0.9843
Epoch 12/15
3251/3251 [==============================] - 14s 4ms/step - loss: 0.0523 - accuracy: 0.9840
Epoch 13/15
3251/3251 [==============================] - 14s 4ms/step - loss: 0.0504 - accuracy: 0.9847
Epoch 14/15
3251/3251 [==============================] - 16s 5ms/step - loss: 0.0528 - accuracy: 0.9842
Epoch 15/15
3251/3251 [==============================] - 15s 5ms/step - loss: 0.0516 - accuracy: 0.9851
```

Model Summery and Performance

Evaluate Model

```
model3.evaluate(x_test,y_test)
```

```
5572/5572 [==============================] - 16s 3ms/step - loss: 0.0148 - accuracy: 0.9953
[0.014784570783376694, 0.9953336119651794]
```

Model Summary

```
model3.summary()
```

```
Model: "sequential_18"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_72 (Dense)            (None, 128)               1024

 dropout_32 (Dropout)        (None, 128)               0

 dense_73 (Dense)            (None, 64)                8256

 dropout_33 (Dropout)        (None, 64)                0

 dense_74 (Dense)            (None, 32)                2080

 dropout_34 (Dropout)        (None, 32)                0

 dense_75 (Dense)            (None, 16)                528

 dropout_35 (Dropout)        (None, 16)                0

 dense_76 (Dense)            (None, 3)                 51

=================================================================
Total params: 11939 (46.64 KB)
Trainable params: 11939 (46.64 KB)
Non-trainable params: 0 (0.00 Byte)
```

Precision

```
from sklearn.preprocessing import LabelBinarizer
label_binarizer = LabelBinarizer()
y_true_one_hot = label_binarizer.fit_transform(y_test)

# Convert the one-hot encoded true labels to class labels
y_true_classes = tf.argmax(y_true_one_hot, axis=1)

# Precision
precision = tf.keras.metrics.Precision()
precision.update_state(y_true_classes, y_pred)
precision.result()
```

```
<tf.Tensor: shape=(), dtype=float32, numpy=0.9990466>
```

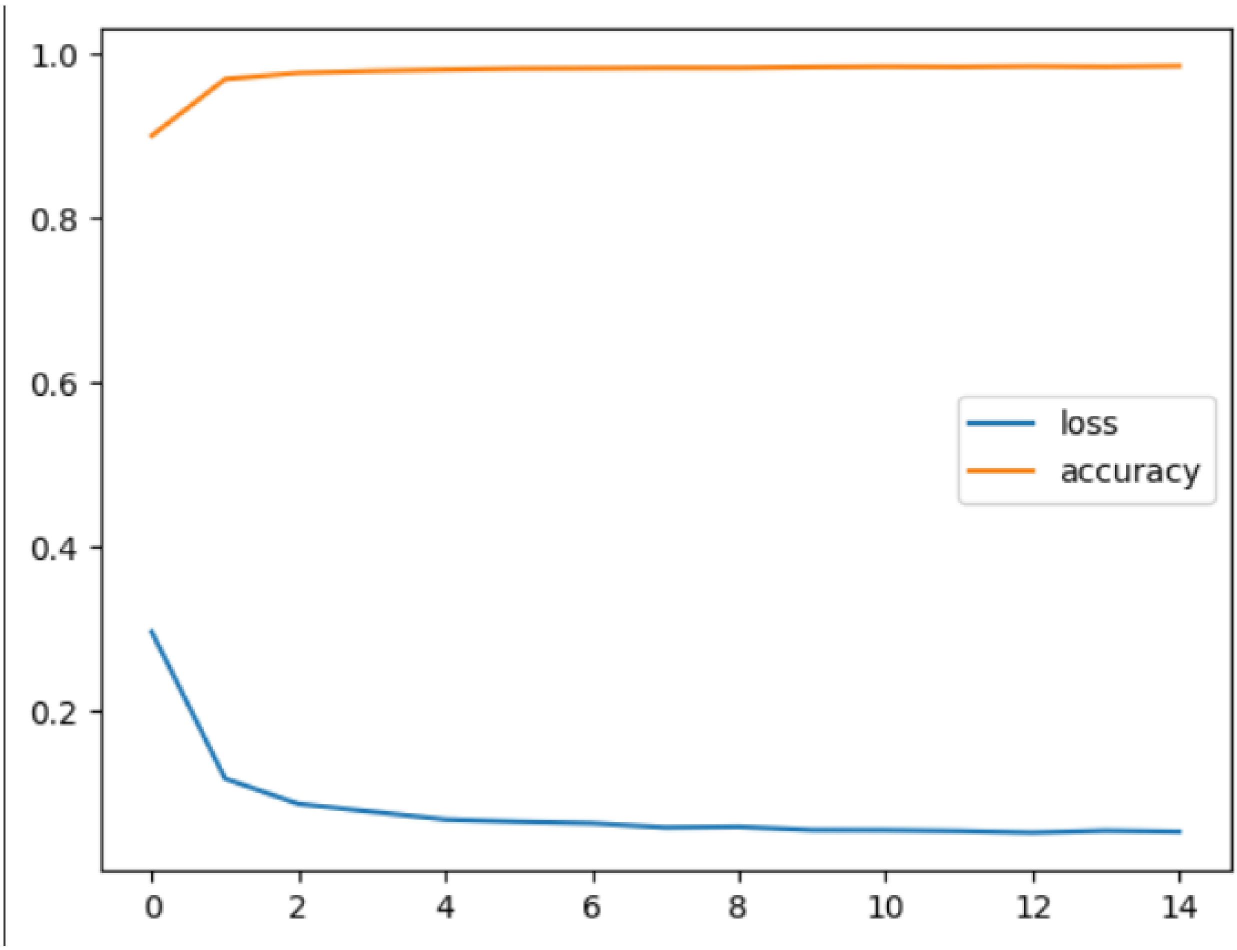
Recall

```
# Recall
recall = tf.keras.metrics.Recall()
recall.update_state(y_true_classes, y_pred)
recall.result()
```

```
<tf.Tensor: shape=(), dtype=float32, numpy=0.99506074>
```

- ✓ Training_Accuracy -> 98.51%
- ✓ Testing_Accuracy -> 99.53%
- ✓ Precision -> 99.90%
- ✓ Recall -> 99.50%
- ✓ No.Epochs -> 15
- ✓ Batch_Size -> 128
- ✓ Learning_Rate -> 0.001
- ✓ Optimizer -> Adam

Training Accuracy and Loss Over Epochs



Model_Architecture_2

```
model4 = tf.keras.Sequential([
    tf.keras.layers.Dense(256, activation='relu',
input_shape=(7,)),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(8, activation='relu'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(3, activation='softmax')
])
```

Model Architecture 2, distinguished by a more intricate structure including an extra Dense layer with 8 neurons and a lower dropout rate of 0.3, demonstrated enhanced capabilities compared to Model Architecture 1. The added complexity in Model Architecture 2 allowed for a more nuanced representation of data patterns, making it a preferable choice for our specific task. The lower dropout rate in Model Architecture 2 contributed to improved information flow during training, highlighting its potential advantages over the simpler Model Architecture 1.

Model_4(Best)

Setting up the layers of Neural Network

```python
model4 = tf.keras.Sequential([
    tf.keras.layers.Dense(256, activation='relu', input_shape=(7,)),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(8, activation='relu'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(3, activation='softmax')
])
```

compile the model

```python
model4.compile(
    loss=tf.keras.losses.categorical_crossentropy,
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
    metrics=['accuracy']
  )
```

Train the Model

```python
epoch_number = 10
history = model4.fit(x_train, y_train, epochs=epoch_number,batch_size=64)
```

```
Epoch 1/10
6501/6501 [==============================] - 36s 5ms/step - loss: 0.1671 - accuracy: 0.9437
Epoch 2/10
6501/6501 [==============================] - 33s 5ms/step - loss: 0.0731 - accuracy: 0.9749
Epoch 3/10
6501/6501 [==============================] - 36s 6ms/step - loss: 0.0641 - accuracy: 0.9773
Epoch 4/10
6501/6501 [==============================] - 32s 5ms/step - loss: 0.0606 - accuracy: 0.9785
Epoch 5/10
6501/6501 [==============================] - 34s 5ms/step - loss: 0.0568 - accuracy: 0.9788
```

```
Epoch 7/10
6501/6501 [==============================] - 32s 5ms/step - loss: 0.0577 - accuracy: 0.9787
Epoch 8/10
6501/6501 [==============================] - 35s 5ms/step - loss: 0.0501 - accuracy: 0.9804
Epoch 9/10
6501/6501 [==============================] - 33s 5ms/step - loss: 0.0525 - accuracy: 0.9801
Epoch 10/10
6501/6501 [==============================] - 32s 5ms/step - loss: 0.0519 - accuracy: 0.9806
```

Model Summery and Performance

```
Model: "sequential_21"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_91 (Dense)            (None, 256)               2048

 dropout_48 (Dropout)        (None, 256)               0

 dense_92 (Dense)            (None, 128)               32896

 dropout_49 (Dropout)        (None, 128)               0

 dense_93 (Dense)            (None, 64)                8256

 dropout_50 (Dropout)        (None, 64)                0

 dense_94 (Dense)            (None, 32)                2080

 dropout_51 (Dropout)        (None, 32)                0

 dense_95 (Dense)            (None, 16)                528

 dropout_52 (Dropout)        (None, 16)                0

 dense_96 (Dense)            (None, 8)                 136

 dropout_53 (Dropout)        (None, 8)                 0

 dense_97 (Dense)            (None, 3)                 27

=================================================================
Total params: 45971 (179.57 KB)
Trainable params: 45971 (179.57 KB)
```

Precision

```python
from sklearn.preprocessing import LabelBinarizer
label_binarizer = LabelBinarizer()
y_true_one_hot = label_binarizer.fit_transform(y_test)

# Convert the one-hot encoded true labels to class labels
y_true_classes = tf.argmax(y_true_one_hot, axis=1)

# Precision
precision = tf.keras.metrics.Precision()
precision.update_state(y_true_classes, y_pred)
precision.result()
```

```
<tf.Tensor: shape=(), dtype=float32, numpy=0.9950652>
```

Recall

```python
recall = tf.keras.metrics.Recall()
recall.update_state(y_true_classes, y_pred)
recall.result()
```

```
<tf.Tensor: shape=(), dtype=float32, numpy=0.99596435>
```

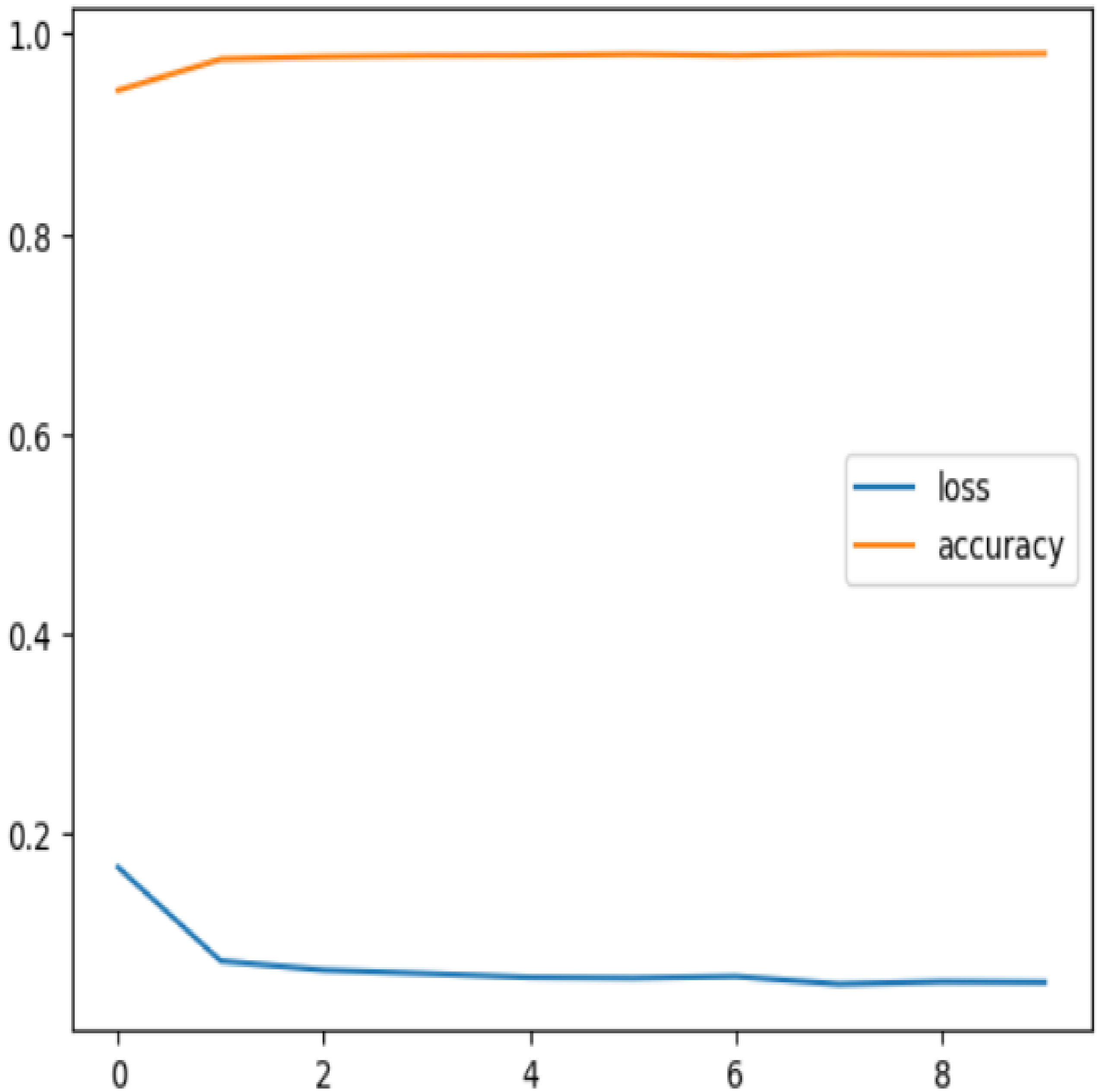
Evaluate Model

```python
model4.evaluate(x_test,y_test)
```

```
5572/5572 [==============================] - 16s 3ms/step - loss: 0.0155 - accuracy: 0.9930
[0.015489117242395878, 0.9930003881454468]
```

- ✓ Training_Accuracy -> 98.06%
- ✓ Testing_Accuracy -> 99.30%
- ✓ Precision -> 99.50%
- ✓ Recall -> 99.59%
- ✓ No.Epochs -> 10
- ✓ Batch_Size -> 64
- ✓ Learning_Rate -> 0.001
- ✓ Optimizer -> Adam

Training Accuracy and Loss Over Epochs



Precision and recall are two metrics used to evaluate the performance of a classification model:

Precision→

- Precision is the ratio of true positive predictions to the total number of positive predictions made by the model.
- Precision = True Positives / (True Positives + False Positives)
- Precision indicates the accuracy of the positive predictions made by the model, emphasizing the reliability of the model when it predicts a positive outcome.

Recall→

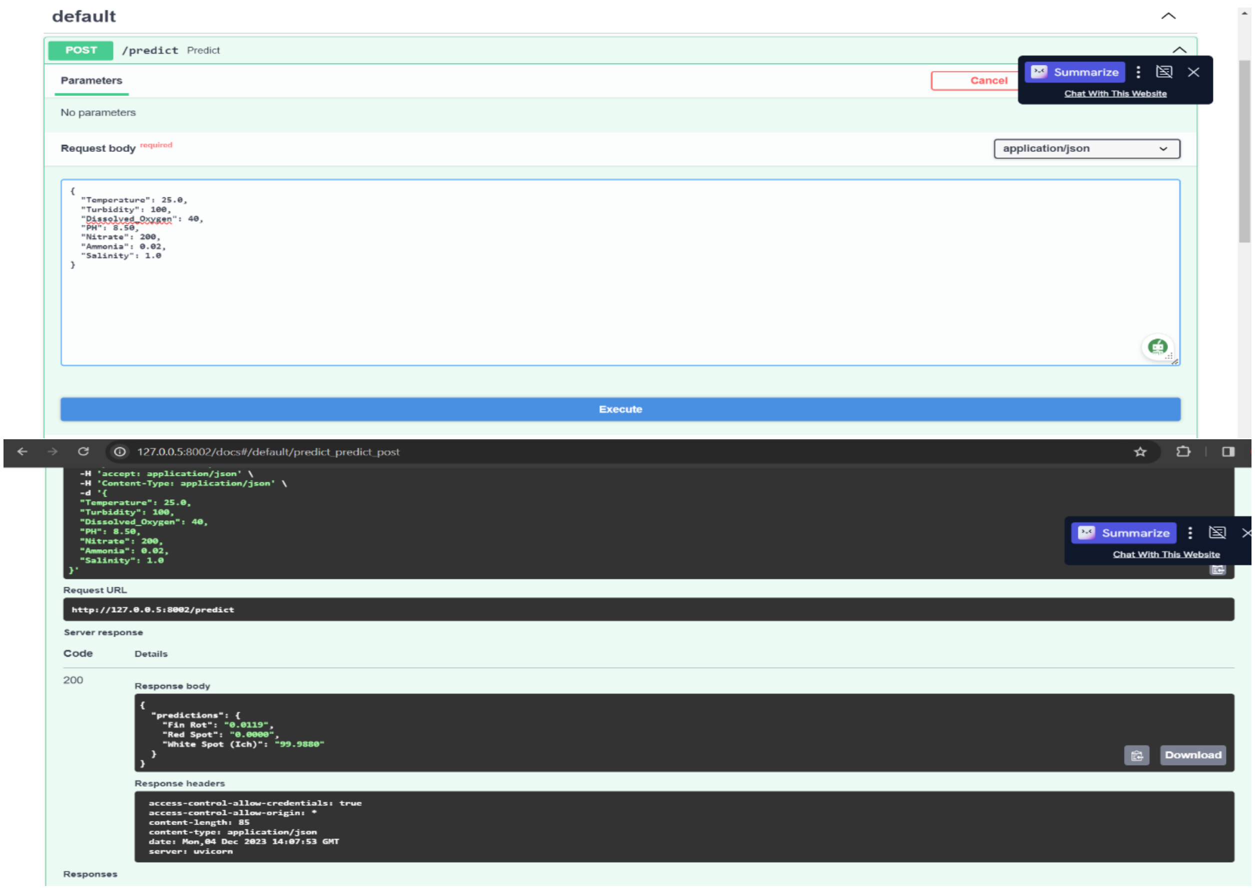
- Recall, is the ratio of true positive predictions to the total number of actual positives in the dataset.
- Recall = True Positives / (True Positives + False Negatives)
- Recall measures the ability of the model to capture and identify all the relevant instances of a positive class, emphasizing the model's sensitivity to positive cases.

Some Predictions Using the Best Model…

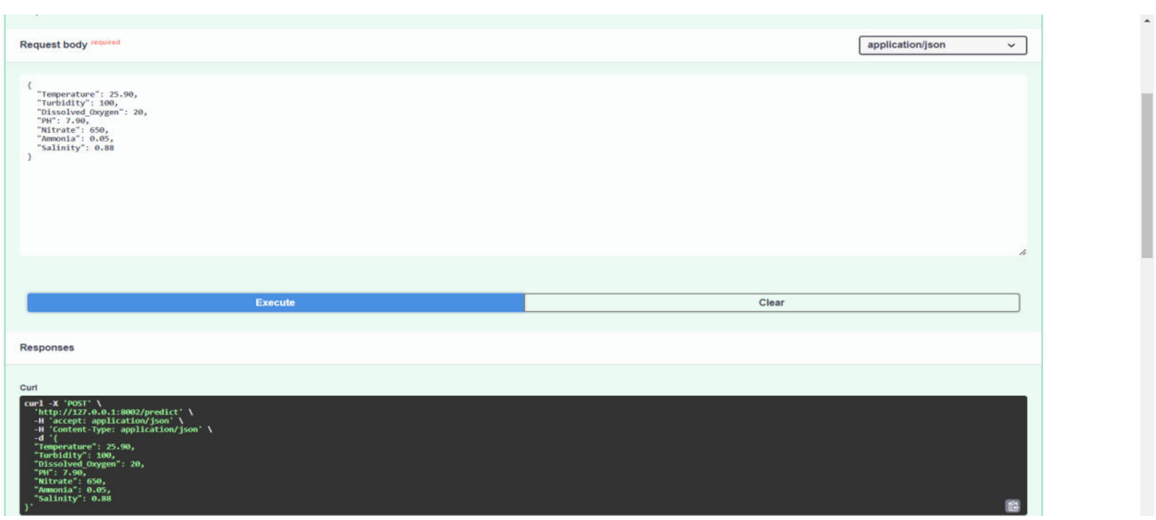
```
import numpy as np
import tensorflow as tf

# Assuming you have a dictionary that maps class indices to their labels
class_labels = {0: "Fin Rot", 1: "Red Spot", 2: "White Spot (Ich)"}

# Load the model
new_model_ = tf.keras.models.load_model('/content/drive/MyDrive/new_model_')

# Make a prediction
prediction = new_model_.predict(new_data_point_scaled)

# Convert the predictions to percentage without rounding
percentage_values = np.round(prediction.flatten() * 100, 4)

# Display predictions with class names
print("Predictions:")
for label, percentage in zip(class_labels.values(), percentage_values):
    print(f"{label}: {percentage:.4f}%")
```

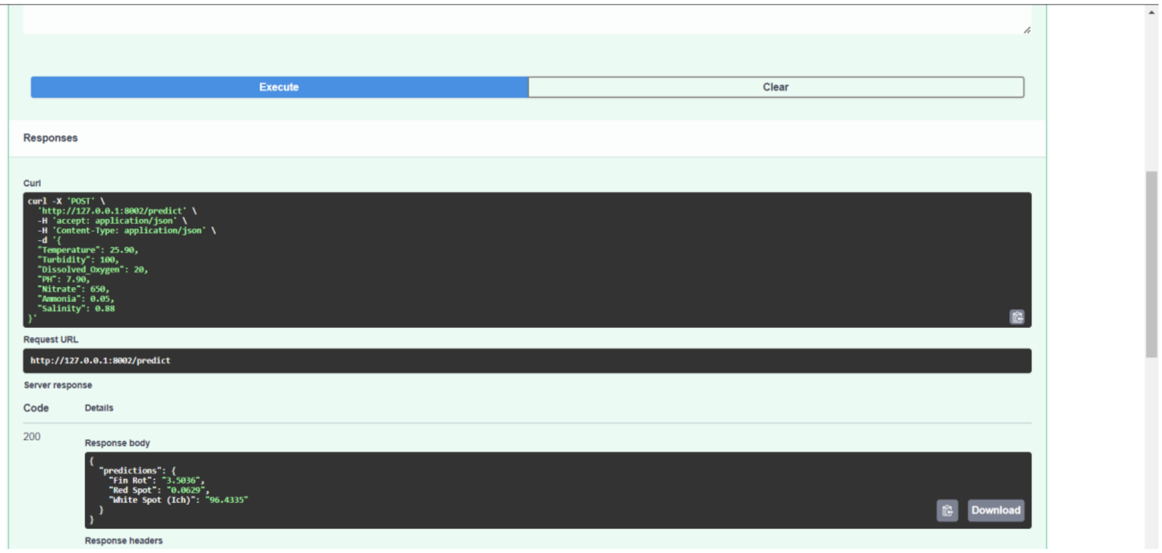
```
WARNING:tensorflow:5 out of the last 8 calls to <function Model.make_predict_function.<locals>.predict_function at 0x7dd4b9730dc0> triggered tf.function retracing. Tracing is expensive
1/1 [==============================] - 0s 72ms/step
Predictions:
Fin Rot: 10.1057%
Red Spot: 89.8943%
White Spot (Ich): 0.0000%
```

Prediction 2

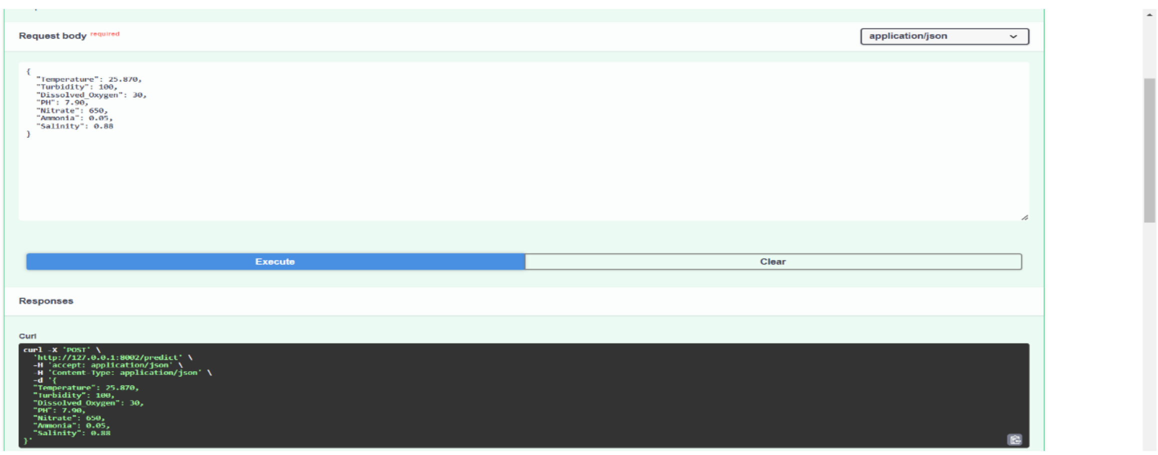


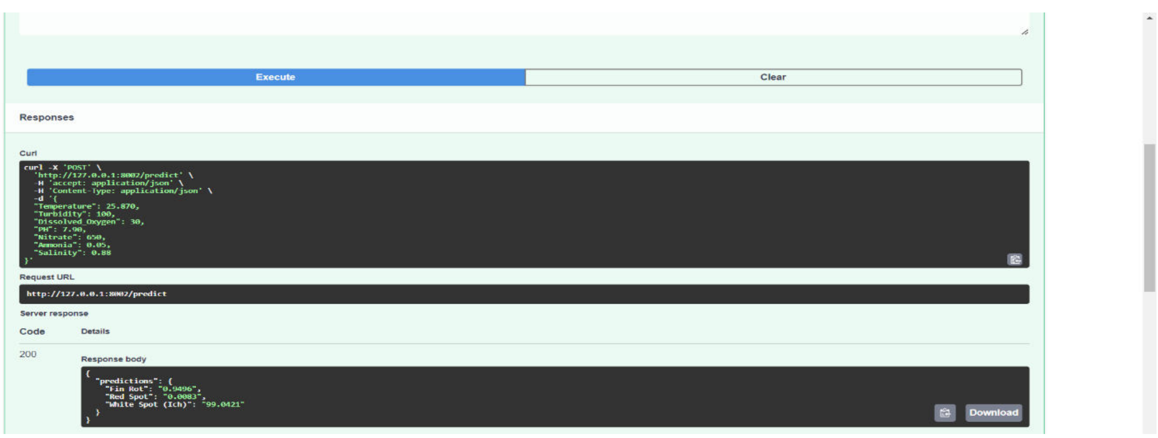
Prediction 3





Prediction 4





Prediction 5