Question 1

```
import numpy as np
import matplotlib.pyplot as plt

class NeuralNetwork:
    class Layer:
        def __init__(self, input_size, output_size, activation, 
wtb_init = 'random'):
            self.vals = None
            self.res = None
            self.activation = activation
            self.weights, self.bias = self.__init_wtb__(input_size, 
output_size, wtb_init)

        def __init_wtb__(self, input_size, output_size, init_mth):
            if init_mth == 'random':
                weights = np.random.rand(input_size, output_size)
                bias = np.random.rand(output_size)
            elif init_mth == 'zeros':
                weights = np.zeros((input_size, output_size))
                bias = np.zeros(output_size)
            elif init_mth == 'normal':
                mean, std_dev = 0, 0.1
                weights = np.random.normal(mean, std_dev, (input_size, 
output_size))
                bias = np.random.normal(mean, std_dev, output_size)
            else:
                raise ValueError("Invalid initialization method. Use 
'random', 'zeros', or 'normal'.")
            return weights, bias

        def __get_act__(self, x):
            if self.activation == 'sigmoid':
                return 1 / (1 + np.exp(-x))
            elif self.activation == 'relu':
                return np.maximum(0, x)
            elif self.activation == 'tanh':
                return np.tanh(x)
            elif self.activation == 'linear':
                return x

        def __get_act_grad__(self, x):
            if self.activation == 'sigmoid':
                z = 1 / (1 + np.exp(-x))
                return z * (1 - z)
            elif self.activation == 'relu':
                return np.where(x > 0, 1, 0)
```

```
            elif self.activation == 'tanh':
                return 1-np.tanh(x)**2
            elif self.activation == 'linear':
                return np.ones_like(x)

        def forward(self, vals):
            self.vals = vals
            self.res = np.dot(self.vals, self.weights) + self.bias
            return self.__get_act__(self.res)

        def backward(self, err, lr):
            grad_x = self.__get_act_grad__(self.res) * err
            nex_err = np.dot(grad_x, self.weights.T)
            self.weights = self.weights - lr * np.dot(self.vals.T,
err)
            self.bias = self.bias - lr * err
            return nex_err

    def __init__(self):
        self.layers = []

    def __loss_func(self,y_true, y_pred, loss ='mse'):
        n = len(y_true)
        diff = y_true - y_pred

        if loss == 'mse':
            err = np.mean(diff**2)
        elif loss == 'cross_entropy':
            eps = 1e-15
            clpd_preds = np.clip(y_pred, eps, 1 - eps)
            err = -np.sum(y_true * np.log(clpd_preds)) / n
        else:
            raise ValueError("Invalid loss type. Use 'mse' or
'cross_entropy'.")

        return err

    def __grad_loss(self, y_true, y_pred, loss='mse'):
        n = len(y_true)
        diff = y_true - y_pred

        if loss == 'mse':
            grad_err = -2 * diff / n
        elif loss == 'cross_entropy':
            eps = 1e-15
            clpd_preds = np.clip(y_pred, eps, 1 - eps)
            grad_err = -y_true / (clpd_preds * n)
        else:
            raise ValueError("Invalid loss type. Use 'mse' or
'cross_entropy'.")
```

```
        return grad_err


    def add(self, input_size, output_size, activation):
        self.layers.append(self.Layer(input_size = input_size,
output_size = output_size, activation = activation))

    def __f_prop__(self, input):
        res = input
        for layer in self.layers:
            res = layer.forward(res)
        return res
    def __b_prop__(self, err, lr):
        res = err
        rev_lyr = reversed(self.layers)
        for layer in rev_lyr:
            res = layer.backward(res, lr)
        return res

    def print_weights(self, epoch):
        print(f"\n{'=' * 30} Epoch {epoch + 1} {'=' * 30}")
        for i, layer in enumerate(self.layers):
            print(f"Layer {i + 1} Weights:")
            for k in range(layer.weights.shape[0]):
                print(f"  Neuron_{k + 1:<3}", end="")
                for w in layer.weights[k]:
                    print(f"  {w:.4f}  ", end="")
                print()

        print("-" * 50)

    def predict(self, X):
        predictions = []
        for sample in range(X.shape[0]):
            pred = self.__f_prop__(X[sample])
            predictions.append(pred)
        return np.array(predictions)

    def score(self, X, y):
        predictions = self.predict(X)
        accuracy = np.sum(np.round(predictions) == y) / len(y)
        return accuracy

    def fit(self, X_train, y_train, X_val, y_val, epochs,
learning_rate, plot=False, verbose=False, accuracy=True):
        samples = X_train.shape[0]
        train_errors = []
        val_errors = []
        val_accuracies = []
```

```
        train_accuracies = []

        for epoch in range(epochs):
            err = 0
            val_err = 0

            for sample in range(samples):
                curr = X_train[sample]
                pred = self.__f_prop__(curr)
                err += self.__loss_func(y_train[sample], pred)

                in_err = self.__grad_loss(y_train[sample], pred)
                self.__b_prop__(in_err, learning_rate)

            for val_sample in range(X_val.shape[0]):
                val_pred = self.__f_prop__(X_val[val_sample])
                val_err += self.__loss_func(y_val[val_sample],
val_pred)

            err /= samples
            val_err /= X_val.shape[0]

            train_errors.append(err)
            val_errors.append(val_err)

            if accuracy:
                val_accuracy = self.score(X_val, y_val)
                train_accuracy = self.score(X_train, y_train)
                val_accuracies.append(val_accuracy)
                train_accuracies.append(train_accuracy)

            if verbose:
                self.print_weights(epoch)
                print(f'Training Error: {err:.4f}   Validation Error:
{val_err:.4f}\n')

        if plot:
            plt.plot(range(1, epochs + 1), train_errors,
label='Training Loss', color='blue')
            plt.plot(range(1, epochs + 1), val_errors,
label='Validation Loss', color='green')
            plt.xlabel('Epochs')
            plt.ylabel('Loss')
            plt.legend()
            plt.show()

        if accuracy:
            plt.plot(range(1, epochs + 1), val_accuracies,
label='Validation Accuracy', color='red')
            plt.plot(range(1, epochs + 1), train_accuracies,
```

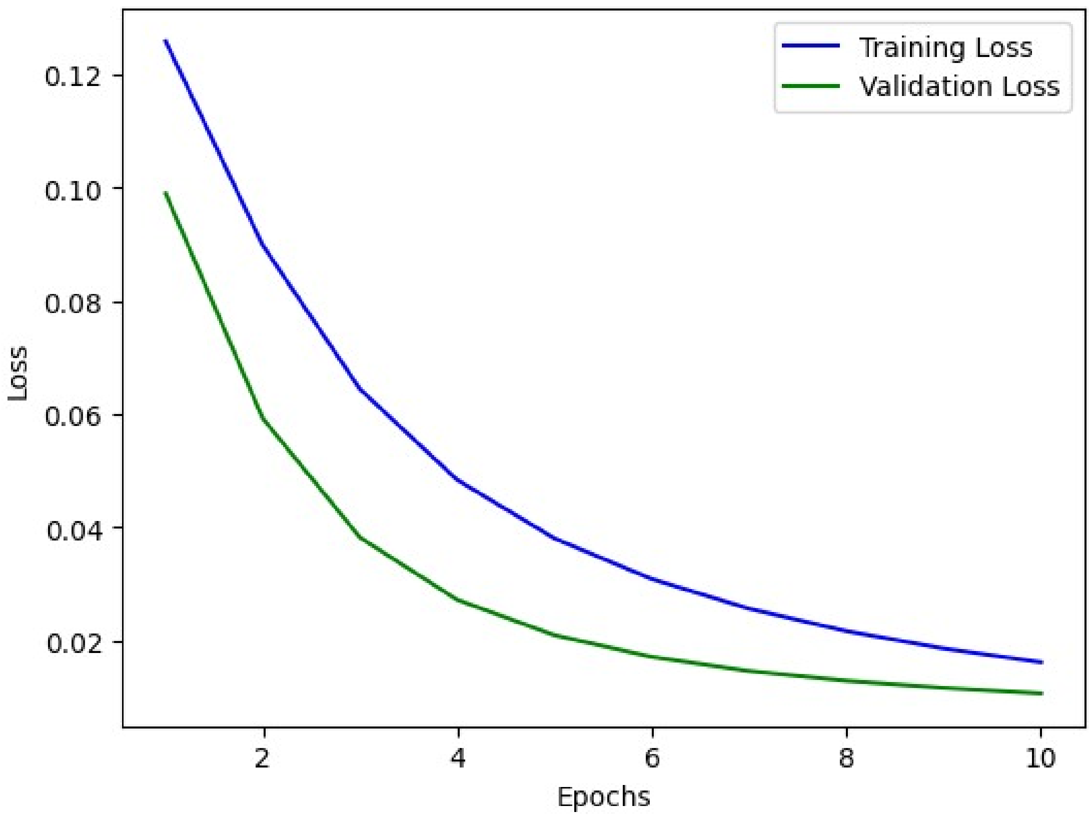
```
label='Test Accuracy', color='purple')
            plt.xlabel('Epochs')
            plt.ylabel('Accuracy')
            plt.legend()
            plt.show()
        print(f'Training Accuracy: {self.score(X_train, y_train):.4f}
Validation Accuracy: {self.score(X_val, y_val):.4f}\n')

from sklearn.model_selection import train_test_split

np.random.seed(42)

X = np.random.randint(-5, 5, size=(100, 1, 2))
y = np.array([[[0]] if x[0, 0] < 0 else [[1]] for x in X])

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.3,
random_state=42)

model = NeuralNetwork()
model.add(input_size=2, output_size=4, activation="linear")
model.add(input_size=4, output_size=1, activation="sigmoid")
model.fit(X_train, y_train, X_val, y_val, epochs = 10, learning_rate=
0.01, plot= True, verbose=True, accuracy=True)
```

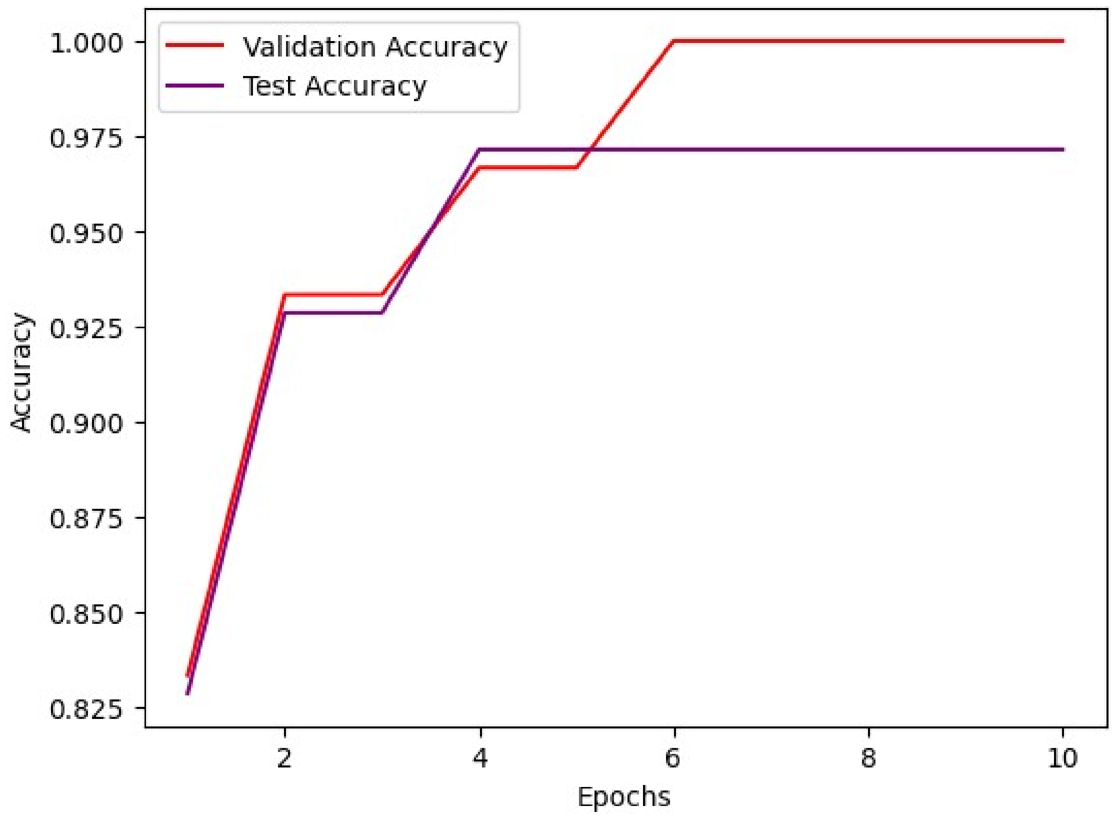
```
============================== Epoch 1 ==============================
Layer 1 Weights:
  Neuron_1    0.4011    0.8976    0.8238    0.1899
  Neuron_2    0.5065    0.6048    0.8309    0.2832
Layer 2 Weights:
  Neuron_1    -0.0573
  Neuron_2    0.8848
  Neuron_3    0.1781
  Neuron_4    0.3625
--------------------------------------------------
Training Error: 0.1258   Validation Error: 0.0990


============================== Epoch 2 ==============================
Layer 1 Weights:
  Neuron_1    0.3922    0.9746    0.8334    0.2169
  Neuron_2    0.5157    0.5233    0.8206    0.2554
Layer 2 Weights:
  Neuron_1    -0.1526
  Neuron_2    0.9616
  Neuron_3    0.0667
  Neuron_4    0.2852
--------------------------------------------------
Training Error: 0.0898   Validation Error: 0.0592
```

```
============================== Epoch 3 ==============================
Layer 1 Weights:
  Neuron_1    0.3810    1.0338    0.8345    0.2322
  Neuron_2    0.5268    0.4620    0.8187    0.2396
Layer 2 Weights:
  Neuron_1    -0.2193
  Neuron_2    1.0839
  Neuron_3    0.0042
  Neuron_4    0.2479
--------------------------------------------------
Training Error: 0.0644   Validation Error: 0.0382


============================== Epoch 4 ==============================
Layer 1 Weights:
  Neuron_1    0.3711    1.0776    0.8331    0.2412
  Neuron_2    0.5368    0.4151    0.8193    0.2298
Layer 2 Weights:
  Neuron_1    -0.2695
  Neuron_2    1.2053
  Neuron_3    -0.0365
  Neuron_4    0.2283
--------------------------------------------------
Training Error: 0.0483   Validation Error: 0.0272


============================== Epoch 5 ==============================
Layer 1 Weights:
  Neuron_1    0.3628    1.1121    0.8311    0.2472
  Neuron_2    0.5456    0.3759    0.8207    0.2228
Layer 2 Weights:
  Neuron_1    -0.3066
  Neuron_2    1.3157
  Neuron_3    -0.0622
  Neuron_4    0.2180
--------------------------------------------------
Training Error: 0.0380   Validation Error: 0.0209


============================== Epoch 6 ==============================
Layer 1 Weights:
  Neuron_1    0.3557    1.1408    0.8291    0.2516
  Neuron_2    0.5535    0.3415    0.8222    0.2172
Layer 2 Weights:
  Neuron_1    -0.3328
  Neuron_2    1.4148
  Neuron_3    -0.0757
  Neuron_4    0.2135
--------------------------------------------------
Training Error: 0.0308   Validation Error: 0.0171
```

```
============================== Epoch 7 ==============================
Layer 1 Weights:
  Neuron_1    0.3497    1.1654    0.8274    0.2551
  Neuron_2    0.5606    0.3106    0.8237    0.2126
Layer 2 Weights:
  Neuron_1    -0.3503
  Neuron_2    1.5039
  Neuron_3    -0.0796
  Neuron_4    0.2129
--------------------------------------------------
Training Error: 0.0256   Validation Error: 0.0146


============================== Epoch 8 ==============================
Layer 1 Weights:
  Neuron_1    0.3445    1.1871    0.8260    0.2581
  Neuron_2    0.5669    0.2828    0.8249    0.2086
Layer 2 Weights:
  Neuron_1    -0.3613
  Neuron_2    1.5844
  Neuron_3    -0.0762
  Neuron_4    0.2149
--------------------------------------------------
Training Error: 0.0217   Validation Error: 0.0129


============================== Epoch 9 ==============================
Layer 1 Weights:
  Neuron_1    0.3399    1.2066    0.8249    0.2607
  Neuron_2    0.5724    0.2576    0.8259    0.2052
Layer 2 Weights:
  Neuron_1    -0.3675
  Neuron_2    1.6575
  Neuron_3    -0.0677
  Neuron_4    0.2185
--------------------------------------------------
Training Error: 0.0186   Validation Error: 0.0116


============================== Epoch 10 ==============================
Layer 1 Weights:
  Neuron_1    0.3359    1.2243    0.8241    0.2629
  Neuron_2    0.5773    0.2349    0.8266    0.2021
Layer 2 Weights:
  Neuron_1    -0.3702
  Neuron_2    1.7242
  Neuron_3    -0.0558
  Neuron_4    0.2232
```

```
----------------------------------------------------------
Training Error: 0.0162   Validation Error: 0.0106
```






```
Training Accuracy: 0.9714   Validation Accuracy: 1.0000
```

Report

Dataset

The dataset consists of 100 points, each with 2 coordinates generated randomly between -5 and 5. The labels are binary, assigned based on whether the x-coordinate is less than 0.

Analysis

The model demonstrates a consistent decrease in both training and validation losses, suggesting effective learning during backpropagation. Training and validation accuracies steadily increase, reaching 97.14% and 100%, respectively.

Explanation

The observed trends reveal the model's ability to fine-tune weights, capturing intricate patterns. As epochs progress, the diminishing rate of loss reduction suggests approaching convergence, while rising accuracies affirm the model's capacity to generalize well to both training and validation datasets, yielding high accuracy.

Question 2

```
import pandas as pd
import matplotlib.pyplot as plt

X = pd.read_csv("gene_exp_X")
X.set_index('Gene Accession Number', inplace=True)
X
```

```
                        AFFX-BioB-5_at  AFFX-BioB-M_at  AFFX-BioB-3_at
\
Gene Accession Number

1                                 -214            -153             -58

2                                 -139             -73              -1

3                                  -76             -49            -307

4                                 -135            -114             265

5                                 -106            -125             -76

...                                ...             ...             ...

68                                -154            -136              49

69                                 -79            -118             -30

70                                 -55             -44              12

71                                 -59            -114              23

72                                -131            -126             -50


                        AFFX-BioC-5_at  AFFX-BioC-3_at  AFFX-BioDn-5_at
\
Gene Accession Number

1                                   88            -295             -558

2                                  283            -264             -400

3                                  309            -376             -650

4                                   12            -419             -585
```

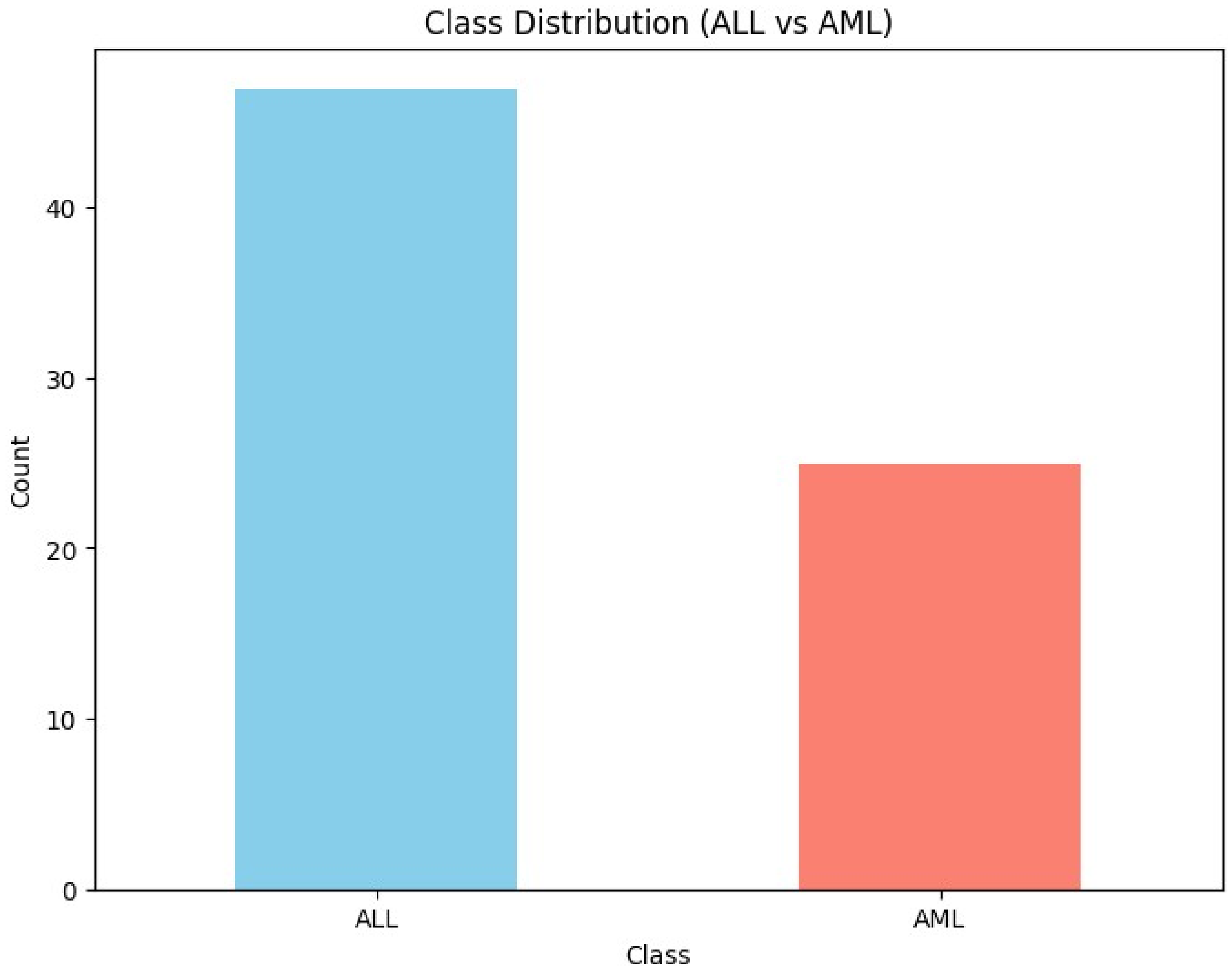
```
5                                     168             -230              -284
...                                   ...              ...               ...
68                                    180             -257              -273
69                                     68             -110              -264
70                                    129             -108              -301
71                                    146             -171              -227
72                                    211             -206              -287

                       AFFX-BioDn-3_at  AFFX-CreX-5_at  AFFX-CreX-3_at  \
Gene Accession Number
1                                  199            -176             252
2                                 -330            -168             101
3                                   33            -367             206
4                                  158            -253              49
5                                    4            -122              70
...                                ...             ...             ...
68                                 141            -123              52
69                                 -28             -61              40
70                                -222            -133             136
71                                 -73            -126              -6
72                                 -34            -114              62

                       AFFX-BioB-5_st  ...  U48730_at  U58516_at
U73738_at  \
Gene Accession Number                  ...
1                                 206  ...        185        511
-125
2                                  74  ...        169        837
-36
```

```
3                                          -215  ...         315       1199
33
4                                            31  ...         240        835
218
5                                           252  ...         156        649
57
...                                         ...  ...         ...        ...
...
68                                          878  ...         214        540
13
69                                         -217  ...         409        617
-34
70                                          320  ...         131        318
35
71                                          149  ...         214        760
-38
72                                          341  ...         206        697
3

                       X06956_at  X16699_at  X83863_at  Z17240_at  \
Gene Accession Number
1                            389        -37        793        329
2                            442        -17        782        295
3                            168         52       1138        777
4                            174       -110        627        170
5                            504        -26        250        314
...                          ...        ...        ...        ...
68                          1075        -45        524        249
69                           738         11        742        234
70                           241        -66        320        174
71                           201        -55        348        208
72                          1046         27        874        393

                       L49218_f_at  M71243_f_at  Z78285_f_at
Gene Accession Number
1                               36          191          -37
2                               11           76          -14
3                               41          228          -41
4                              -50          126          -91
5                               14           56          -25
...                            ...          ...          ...
68                              40          -68           -1
69                              72          109          -30
70                              -4          176           40
71                               0           74          -12
72                              34          237           -2

[72 rows x 7129 columns]
```

```python
y = pd.read_csv("gene_exp_y")
y.set_index('patient', inplace=True)
y
```

```
         cancer
patient
1             0
2             0
3             0
4             0
5             0
...         ...
68            0
69            0
70            0
71            0
72            0

[72 rows x 1 columns]
```

```python
def class_distro(df):

    plt.figure(figsize=(8, 6))
    df['cancer'].value_counts().plot(kind='bar', color=['skyblue', 
'salmon'])
    plt.title('Class Distribution (ALL vs AML)')
    plt.xlabel('Class')
    plt.ylabel('Count')
    plt.xticks([0, 1], ['ALL', 'AML'], rotation=0)
    plt.show()

    cls_dist= df['cancer'].value_counts()
    print("Class Distribution:")
    print(cls_dist)
    print()
    class_proportions = df.value_counts(normalize=True)
    print("Class Proportions:")
    print(f"Class 0: {class_proportions[0]:.2%}")
    print(f"Class 1: {class_proportions[1]:.2%}")

class_distro(y)
```




```
Class Distribution:
0    47
1    25
Name: cancer, dtype: int64

Class Proportions:
Class 0: 65.28%
Class 1: 34.72%
```

The class distribution provides insights into the proportion of ALL (Class 0) and AML (Class 1) samples, allowing us to identify the imbalanced nature of the dataset.

Evaluation Metric

The F1 score is favored for imbalanced datasets due to its balanced consideration of precision and recall. It provides a robust evaluation, particularly when the minority class is of greater interest, offering a stable metric less influenced by class imbalance, and facilitating threshold adjustments for improved model performance.

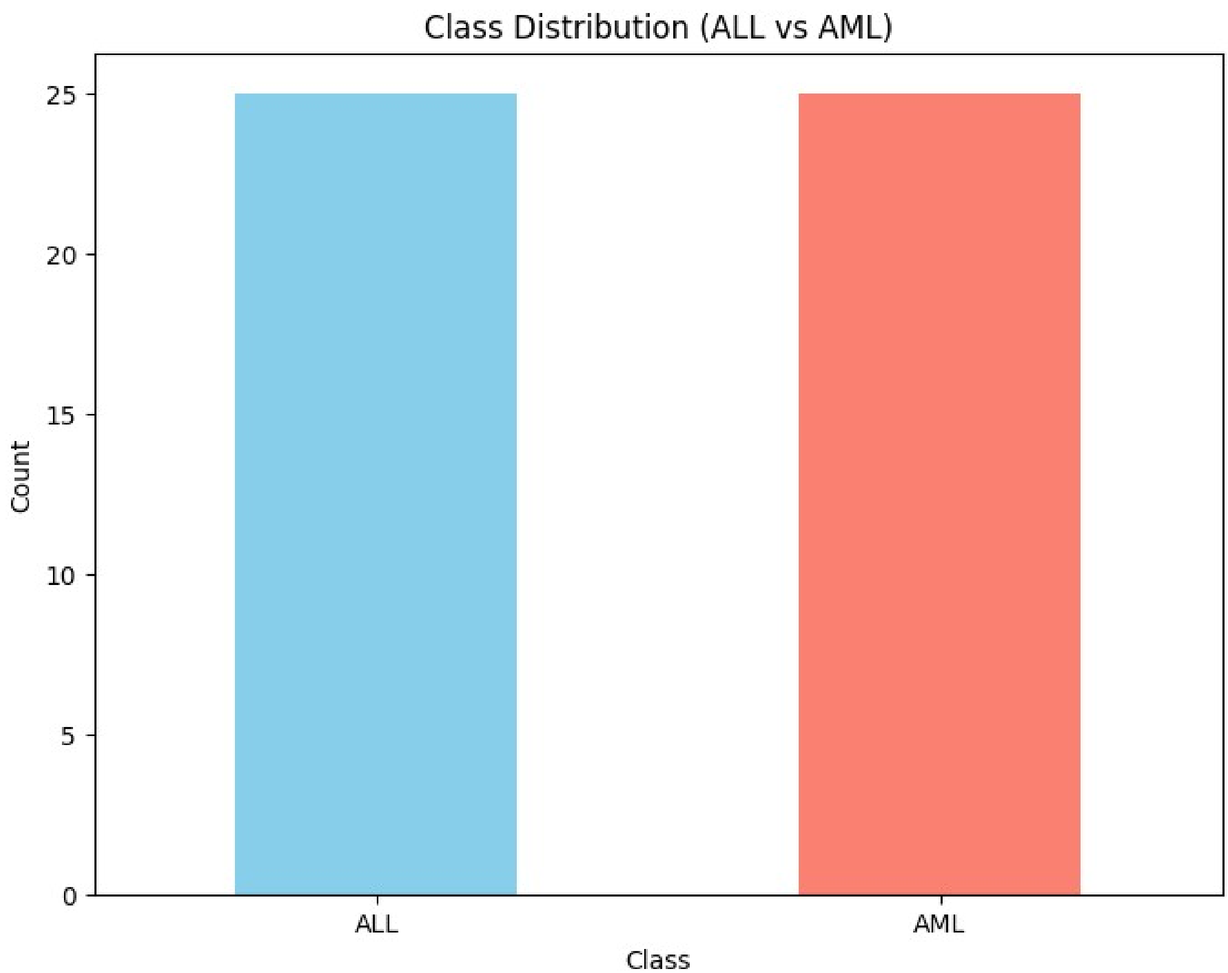
Hence F1 score is used to evaluate the models.

Resample

we can down sample ALL to get a more balanced data

Downsample

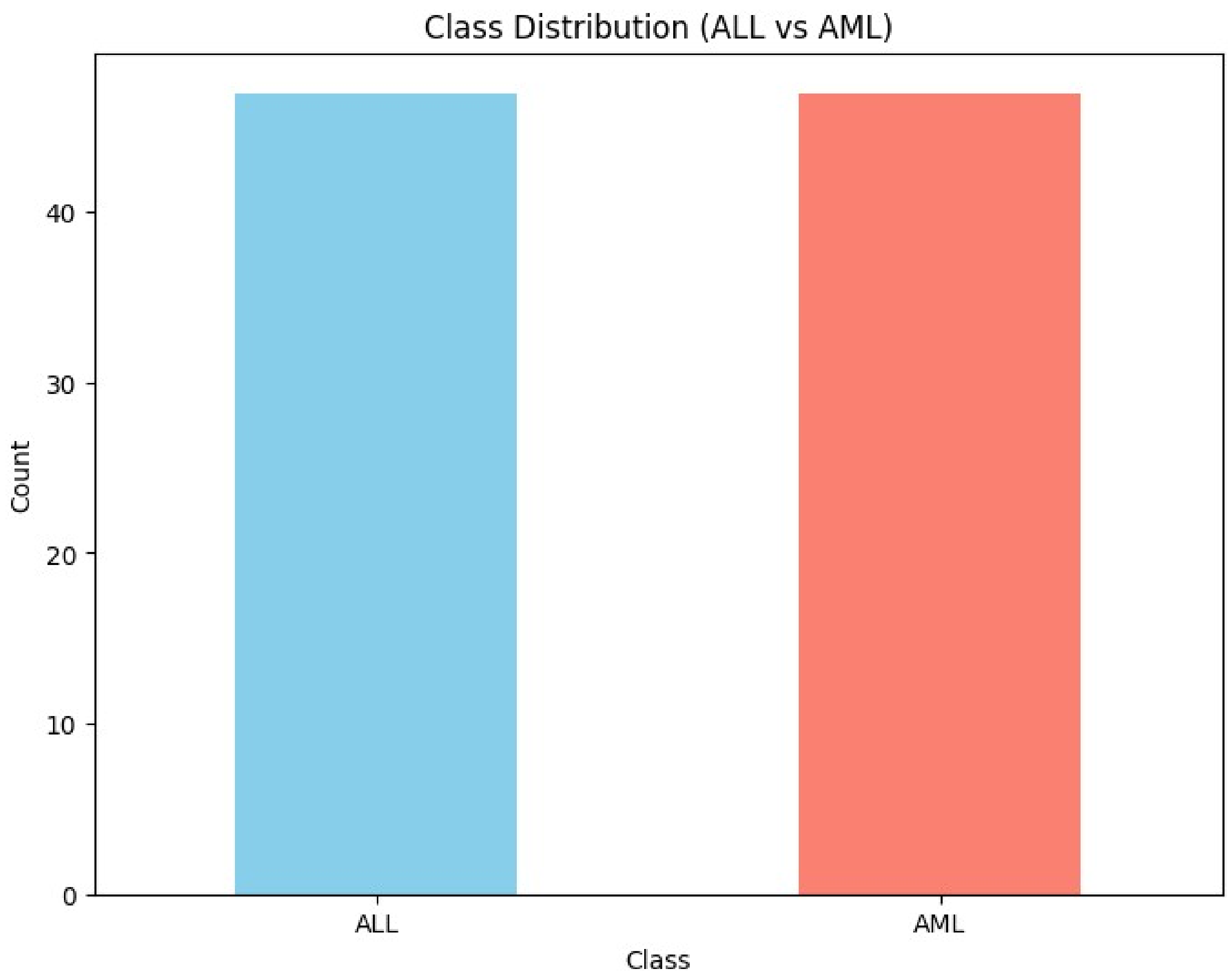
```
from sklearn.utils import resample

def downsample(X, y):
    # Separate the majority and minority classes in y
    mc_y = y[y['cancer'] == 0]
    mnc_y = y[y['cancer'] == 1]

    dwnsmpld_mc_y = resample(mc_y, replace=False,
n_samples=len(mnc_y), random_state=42)

    y_d = pd.concat([dwnsmpld_mc_y, mnc_y])
    X_d = X.loc[y_d.index]

    return X_d, y_d
X_d, y_d = downsample(X, y)

class_distro(y_d)
```




```
Class Distribution:
0    25
1    25
Name: cancer, dtype: int64

Class Proportions:
Class 0: 50.00%
Class 1: 50.00%
```

Upsample

```
from imblearn.over_sampling import SMOTE

# Step 1: Initialize SMOTE
smote = SMOTE(sampling_strategy='auto' ,random_state=42)

# Step 2: Resample the data
X_u, y_u = smote.fit_resample(X, y)

class_distro(y_u)
```




```
Class Distribution:
0    47
1    47
Name: cancer, dtype: int64

Class Proportions:
Class 0: 50.00%
Class 1: 50.00%

import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import f1_score
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

```
# Define custom F1 score metric
class F1Score(tf.keras.metrics.Metric):
    def __init__(self, name='f1_score', **kwargs):
        super(F1Score, self).__init__(name=name, **kwargs)
        self.true_positives = self.add_weight(name='tp',
initializer='zeros')
        self.false_positives = self.add_weight(name='fp',
initializer='zeros')
        self.false_negatives = self.add_weight(name='fn',
initializer='zeros')

    def update_state(self, y_true, y_pred, sample_weight=None):
        y_true = tf.cast(y_true, tf.float32)
        y_pred = tf.cast(tf.math.round(y_pred), tf.float32)

        tp = tf.reduce_sum(y_true * y_pred)
        fp = tf.reduce_sum((1 - y_true) * y_pred)
        fn = tf.reduce_sum(y_true * (1 - y_pred))

        self.true_positives.assign_add(tp)
        self.false_positives.assign_add(fp)
        self.false_negatives.assign_add(fn)

    def result(self):
        precision = self.true_positives / (self.true_positives +
self.false_positives + tf.keras.backend.epsilon())
        recall = self.true_positives / (self.true_positives +
self.false_negatives + tf.keras.backend.epsilon())
        f1 = 2 * (precision * recall) / (precision + recall +
tf.keras.backend.epsilon())
        return f1

# Function to train and plot the neural network
def train_and_plot_nn(X, y, test_size=0.2, random_state=42,
epochs=100, batch_size=64):
    # Standardize the input features
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # Split the data into training and validation sets
    # class_weights = dict(1 /
y_train.astype(int).value_counts(normalize=True))
    X_train, X_val, y_train, y_val = train_test_split(X_scaled, y,
test_size=test_size, random_state=random_state)

    # Calculate class weights based on class distribution
    class_weights = {0: 1.5405405405405406, 1: 2.85}

    # Build the neural network model
    NN_model = keras.Sequential([
```

```
        layers.Dense(64, activation='relu',
input_shape=X_train.shape[1:]),
        layers.Dropout(0.5),
        layers.Dense(32, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(1, activation='sigmoid')
    ])

    # Compile the model with the custom F1 score metric
    NN_model.compile(
        loss='binary_crossentropy',
        optimizer=keras.optimizers.Adam(learning_rate=0.001),
        metrics=[F1Score()]
    )

    # Train the model
    history = NN_model.fit(
        X_train, y_train,
        validation_data=(X_val, y_val), class_weight=class_weights,
        batch_size=batch_size,
        epochs=epochs
    )

    # Plot the training history
    fig, ax = plt.subplots(1, 2, dpi=200, figsize=(4.2, 2.1))
    ax[0].plot(history.history['loss'], label="train loss", lw=0.5)
    ax[0].plot(history.history['val_loss'], label="val loss", lw=0.5)
    ax[1].plot(history.history['f1_score'], label="train F1 score",
lw=0.5)
    ax[1].plot(history.history['val_f1_score'], label="val F1 score",
lw=0.5)
    ax[0].legend(fontsize=5)
    ax[1].legend(fontsize=5)
    ax[0].set_title("Loss curve", fontsize=7)
    ax[1].set_title("F1 Score curve", fontsize=7)
    ax[0].set_xlabel("epoch", fontsize=5)
    ax[1].set_xlabel("epoch", fontsize=5)
    ax[0].set_ylabel("loss", fontsize=5)
    ax[1].set_ylabel("F1 Score", fontsize=5)
    ax[0].tick_params(axis='both', which='major', labelsize=5)
    ax[1].tick_params(axis='both', which='major', labelsize=5)
    plt.suptitle("Neural Network Training with F1 Score", fontsize=8)
    plt.tight_layout()
    plt.show()
```

Original Dataset

```
train_and_plot_nn(X, y)
```

```
Epoch 1/100
1/1 [==============================] - 2s 2s/step - loss: 2.1069 -
f1_score: 0.4706 - val_loss: 0.2667 - val_f1_score: 0.9091
Epoch 2/100
1/1 [==============================] - 0s 94ms/step - loss: 1.2383 -
f1_score: 0.6415 - val_loss: 0.1973 - val_f1_score: 0.9091
Epoch 3/100
1/1 [==============================] - 0s 128ms/step - loss: 0.7729 -
f1_score: 0.7200 - val_loss: 0.1542 - val_f1_score: 0.9091
Epoch 4/100
1/1 [==============================] - 0s 83ms/step - loss: 0.6942 -
f1_score: 0.8372 - val_loss: 0.1241 - val_f1_score: 0.9091
Epoch 5/100
1/1 [==============================] - 0s 121ms/step - loss: 0.9748 -
f1_score: 0.7317 - val_loss: 0.1196 - val_f1_score: 0.9091
Epoch 6/100
1/1 [==============================] - 0s 92ms/step - loss: 1.1081 -
f1_score: 0.7755 - val_loss: 0.1243 - val_f1_score: 0.9091
Epoch 7/100
1/1 [==============================] - 0s 122ms/step - loss: 0.5649 -
f1_score: 0.8163 - val_loss: 0.1348 - val_f1_score: 0.9091
Epoch 8/100
1/1 [==============================] - 0s 162ms/step - loss: 1.0148 -
f1_score: 0.7273 - val_loss: 0.1383 - val_f1_score: 0.9091
Epoch 9/100
1/1 [==============================] - 0s 72ms/step - loss: 0.7887 -
f1_score: 0.8095 - val_loss: 0.1436 - val_f1_score: 0.9091
Epoch 10/100
1/1 [==============================] - 0s 84ms/step - loss: 0.7807 -
f1_score: 0.8182 - val_loss: 0.1378 - val_f1_score: 0.9091
Epoch 11/100
1/1 [==============================] - 0s 90ms/step - loss: 0.4707 -
f1_score: 0.8780 - val_loss: 0.1333 - val_f1_score: 0.9091
Epoch 12/100
1/1 [==============================] - 0s 82ms/step - loss: 0.2133 -
f1_score: 0.9524 - val_loss: 0.1261 - val_f1_score: 0.9091
Epoch 13/100
1/1 [==============================] - 0s 83ms/step - loss: 1.0675 -
f1_score: 0.7826 - val_loss: 0.1201 - val_f1_score: 0.9091
Epoch 14/100
1/1 [==============================] - 0s 173ms/step - loss: 0.3186 -
f1_score: 0.9091 - val_loss: 0.1173 - val_f1_score: 0.9091
Epoch 15/100
1/1 [==============================] - 0s 143ms/step - loss: 0.5069 -
f1_score: 0.9048 - val_loss: 0.1195 - val_f1_score: 0.9091
Epoch 16/100
1/1 [==============================] - 0s 105ms/step - loss: 0.4444 -
f1_score: 0.9500 - val_loss: 0.1228 - val_f1_score: 0.9091
Epoch 17/100
1/1 [==============================] - 0s 85ms/step - loss: 0.7602 -
```

```
f1_score: 0.8571 - val_loss: 0.1318 - val_f1_score: 0.9091
Epoch 18/100
1/1 [==============================] - 0s 95ms/step - loss: 0.6357 -
f1_score: 0.8780 - val_loss: 0.1516 - val_f1_score: 0.9091
Epoch 19/100
1/1 [==============================] - 0s 81ms/step - loss: 0.4348 -
f1_score: 0.8444 - val_loss: 0.1671 - val_f1_score: 0.9091
Epoch 20/100
1/1 [==============================] - 0s 98ms/step - loss: 0.2177 -
f1_score: 0.9500 - val_loss: 0.1759 - val_f1_score: 0.9091
Epoch 21/100
1/1 [==============================] - 0s 75ms/step - loss: 0.7061 -
f1_score: 0.8780 - val_loss: 0.1896 - val_f1_score: 0.9091
Epoch 22/100
1/1 [==============================] - 0s 83ms/step - loss: 0.1392 -
f1_score: 0.9756 - val_loss: 0.1972 - val_f1_score: 0.9091
Epoch 23/100
1/1 [==============================] - 0s 178ms/step - loss: 0.2252 -
f1_score: 0.9500 - val_loss: 0.2009 - val_f1_score: 0.9091
Epoch 24/100
1/1 [==============================] - 0s 80ms/step - loss: 0.3241 -
f1_score: 0.9048 - val_loss: 0.2015 - val_f1_score: 0.9091
Epoch 25/100
1/1 [==============================] - 0s 124ms/step - loss: 0.2545 -
f1_score: 0.9231 - val_loss: 0.1995 - val_f1_score: 0.9091
Epoch 26/100
1/1 [==============================] - 0s 178ms/step - loss: 0.3739 -
f1_score: 0.9048 - val_loss: 0.2002 - val_f1_score: 0.9091
Epoch 27/100
1/1 [==============================] - 0s 80ms/step - loss: 0.2511 -
f1_score: 0.9091 - val_loss: 0.2019 - val_f1_score: 0.9091
Epoch 28/100
1/1 [==============================] - 0s 84ms/step - loss: 0.0057 -
f1_score: 1.0000 - val_loss: 0.2061 - val_f1_score: 0.9091
Epoch 29/100
1/1 [==============================] - 0s 135ms/step - loss: 0.2690 -
f1_score: 0.9302 - val_loss: 0.2095 - val_f1_score: 0.9091
Epoch 30/100
1/1 [==============================] - 0s 117ms/step - loss: 0.3622 -
f1_score: 0.9268 - val_loss: 0.2080 - val_f1_score: 0.9091
Epoch 31/100
1/1 [==============================] - 0s 88ms/step - loss: 0.3837 -
f1_score: 0.9048 - val_loss: 0.2046 - val_f1_score: 0.9091
Epoch 32/100
1/1 [==============================] - 0s 90ms/step - loss: 0.5122 -
f1_score: 0.9744 - val_loss: 0.1950 - val_f1_score: 0.9091
Epoch 33/100
1/1 [==============================] - 0s 89ms/step - loss: 0.0380 -
f1_score: 0.9756 - val_loss: 0.1929 - val_f1_score: 0.9091
Epoch 34/100
```

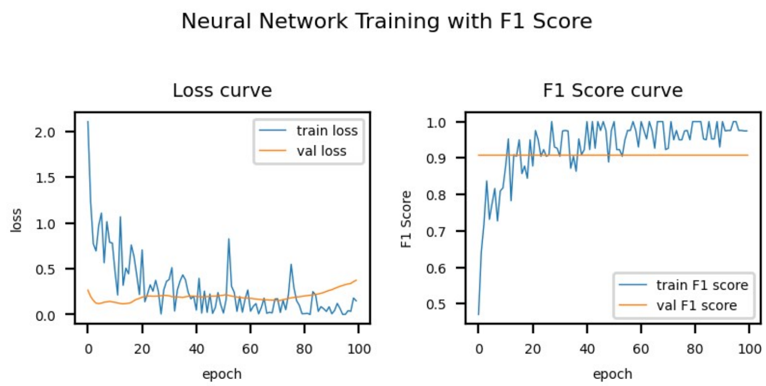
```
1/1 [==============================] - 0s 123ms/step - loss: 0.2669 -
f1_score: 0.9744 - val_loss: 0.1918 - val_f1_score: 0.9091
Epoch 35/100
1/1 [==============================] - 0s 101ms/step - loss: 0.3651 -
f1_score: 0.8718 - val_loss: 0.1905 - val_f1_score: 0.9091
Epoch 36/100
1/1 [==============================] - 0s 91ms/step - loss: 0.4331 -
f1_score: 0.9048 - val_loss: 0.1870 - val_f1_score: 0.9091
Epoch 37/100
1/1 [==============================] - 0s 91ms/step - loss: 0.3787 -
f1_score: 0.8636 - val_loss: 0.1937 - val_f1_score: 0.9091
Epoch 38/100
1/1 [==============================] - 0s 216ms/step - loss: 0.2407 -
f1_score: 0.9524 - val_loss: 0.1988 - val_f1_score: 0.9091
Epoch 39/100
1/1 [==============================] - 0s 175ms/step - loss: 0.1716 -
f1_score: 0.9091 - val_loss: 0.2011 - val_f1_score: 0.9091
Epoch 40/100
1/1 [==============================] - 0s 213ms/step - loss: 0.2022 -
f1_score: 0.9231 - val_loss: 0.1996 - val_f1_score: 0.9091
Epoch 41/100
1/1 [==============================] - 0s 217ms/step - loss: 0.0482 -
f1_score: 1.0000 - val_loss: 0.1978 - val_f1_score: 0.9091
Epoch 42/100
1/1 [==============================] - 0s 205ms/step - loss: 0.3957 -
f1_score: 0.9231 - val_loss: 0.1950 - val_f1_score: 0.9091
Epoch 43/100
1/1 [==============================] - 0s 156ms/step - loss: 0.0159 -
f1_score: 1.0000 - val_loss: 0.1931 - val_f1_score: 0.9091
Epoch 44/100
1/1 [==============================] - 0s 82ms/step - loss: 0.2558 -
f1_score: 0.9268 - val_loss: 0.1928 - val_f1_score: 0.9091
Epoch 45/100
1/1 [==============================] - 0s 134ms/step - loss: 0.0243 -
f1_score: 1.0000 - val_loss: 0.1957 - val_f1_score: 0.9091
Epoch 46/100
1/1 [==============================] - 0s 102ms/step - loss: 0.2248 -
f1_score: 0.9756 - val_loss: 0.1986 - val_f1_score: 0.9091
Epoch 47/100
1/1 [==============================] - 0s 103ms/step - loss: 0.0126 -
f1_score: 1.0000 - val_loss: 0.2006 - val_f1_score: 0.9091
Epoch 48/100
1/1 [==============================] - 0s 96ms/step - loss: 0.0748 -
f1_score: 0.9756 - val_loss: 0.2003 - val_f1_score: 0.9091
Epoch 49/100
1/1 [==============================] - 0s 93ms/step - loss: 0.2397 -
f1_score: 0.8889 - val_loss: 0.2062 - val_f1_score: 0.9091
Epoch 50/100
1/1 [==============================] - 0s 73ms/step - loss: 0.1073 -
f1_score: 0.9756 - val_loss: 0.2078 - val_f1_score: 0.9091
```

```
Epoch 51/100
1/1 [==============================] - 0s 54ms/step - loss: 0.0165 -
f1_score: 1.0000 - val_loss: 0.2110 - val_f1_score: 0.9091
Epoch 52/100
1/1 [==============================] - 0s 50ms/step - loss: 0.1645 -
f1_score: 0.9231 - val_loss: 0.2181 - val_f1_score: 0.9091
Epoch 53/100
1/1 [==============================] - 0s 47ms/step - loss: 0.8270 -
f1_score: 0.9231 - val_loss: 0.2079 - val_f1_score: 0.9091
Epoch 54/100
1/1 [==============================] - 0s 120ms/step - loss: 0.3076 -
f1_score: 0.9048 - val_loss: 0.2016 - val_f1_score: 0.9091
Epoch 55/100
1/1 [==============================] - 0s 122ms/step - loss: 0.2422 -
f1_score: 0.9500 - val_loss: 0.1958 - val_f1_score: 0.9091
Epoch 56/100
1/1 [==============================] - 0s 72ms/step - loss: 0.0346 -
f1_score: 0.9756 - val_loss: 0.1907 - val_f1_score: 0.9091
Epoch 57/100
1/1 [==============================] - 0s 86ms/step - loss: 0.1972 -
f1_score: 0.9756 - val_loss: 0.1870 - val_f1_score: 0.9091
Epoch 58/100
1/1 [==============================] - 0s 90ms/step - loss: 0.0255 -
f1_score: 1.0000 - val_loss: 0.1832 - val_f1_score: 0.9091
Epoch 59/100
1/1 [==============================] - 0s 102ms/step - loss: 0.1696 -
f1_score: 0.9756 - val_loss: 0.1777 - val_f1_score: 0.9091
Epoch 60/100
1/1 [==============================] - 0s 80ms/step - loss: 0.2676 -
f1_score: 0.9302 - val_loss: 0.1788 - val_f1_score: 0.9091
Epoch 61/100
1/1 [==============================] - 0s 75ms/step - loss: 0.0362 -
f1_score: 1.0000 - val_loss: 0.1777 - val_f1_score: 0.9091
Epoch 62/100
1/1 [==============================] - 0s 90ms/step - loss: 0.0888 -
f1_score: 0.9756 - val_loss: 0.1727 - val_f1_score: 0.9091
Epoch 63/100
1/1 [==============================] - 0s 74ms/step - loss: 0.1203 -
f1_score: 0.9524 - val_loss: 0.1691 - val_f1_score: 0.9091
Epoch 64/100
1/1 [==============================] - 0s 78ms/step - loss: 0.0072 -
f1_score: 1.0000 - val_loss: 0.1660 - val_f1_score: 0.9091
Epoch 65/100
1/1 [==============================] - 0s 70ms/step - loss: 0.0772 -
f1_score: 0.9756 - val_loss: 0.1625 - val_f1_score: 0.9091
Epoch 66/100
1/1 [==============================] - 0s 83ms/step - loss: 0.1789 -
f1_score: 0.9268 - val_loss: 0.1613 - val_f1_score: 0.9091
Epoch 67/100
1/1 [==============================] - 0s 119ms/step - loss: 0.0139 -
```

```
f1_score: 1.0000 - val_loss: 0.1598 - val_f1_score: 0.9091
Epoch 68/100
1/1 [==============================] - 0s 126ms/step - loss: 0.0247 -
f1_score: 1.0000 - val_loss: 0.1581 - val_f1_score: 0.9091
Epoch 69/100
1/1 [==============================] - 0s 117ms/step - loss: 0.0181 -
f1_score: 1.0000 - val_loss: 0.1573 - val_f1_score: 0.9091
Epoch 70/100
1/1 [==============================] - 0s 100ms/step - loss: 0.1689 -
f1_score: 0.9231 - val_loss: 0.1551 - val_f1_score: 0.9091
Epoch 71/100
1/1 [==============================] - 0s 174ms/step - loss: 0.1722 -
f1_score: 0.9268 - val_loss: 0.1544 - val_f1_score: 0.9091
Epoch 72/100
1/1 [==============================] - 0s 121ms/step - loss: 0.0228 -
f1_score: 1.0000 - val_loss: 0.1536 - val_f1_score: 0.9091
Epoch 73/100
1/1 [==============================] - 0s 124ms/step - loss: 0.1566 -
f1_score: 0.9500 - val_loss: 0.1626 - val_f1_score: 0.9091
Epoch 74/100
1/1 [==============================] - 0s 133ms/step - loss: 0.0523 -
f1_score: 0.9756 - val_loss: 0.1690 - val_f1_score: 0.9091
Epoch 75/100
1/1 [==============================] - 0s 111ms/step - loss: 0.2194 -
f1_score: 0.9500 - val_loss: 0.1762 - val_f1_score: 0.9091
Epoch 76/100
1/1 [==============================] - 0s 124ms/step - loss: 0.5491 -
f1_score: 0.9500 - val_loss: 0.1821 - val_f1_score: 0.9091
Epoch 77/100
1/1 [==============================] - 0s 124ms/step - loss: 0.2952 -
f1_score: 0.9744 - val_loss: 0.1863 - val_f1_score: 0.9091
Epoch 78/100
1/1 [==============================] - 0s 125ms/step - loss: 0.1508 -
f1_score: 0.9756 - val_loss: 0.1911 - val_f1_score: 0.9091
Epoch 79/100
1/1 [==============================] - 0s 131ms/step - loss: 0.1010 -
f1_score: 0.9500 - val_loss: 0.1936 - val_f1_score: 0.9091
Epoch 80/100
1/1 [==============================] - 0s 181ms/step - loss: 0.0087 -
f1_score: 1.0000 - val_loss: 0.1989 - val_f1_score: 0.9091
Epoch 81/100
1/1 [==============================] - 0s 120ms/step - loss: 0.0101 -
f1_score: 1.0000 - val_loss: 0.2033 - val_f1_score: 0.9091
Epoch 82/100
1/1 [==============================] - 0s 138ms/step - loss: 0.0133 -
f1_score: 1.0000 - val_loss: 0.2070 - val_f1_score: 0.9091
Epoch 83/100
1/1 [==============================] - 0s 122ms/step - loss: 4.3507e-
04 - f1_score: 1.0000 - val_loss: 0.2103 - val_f1_score: 0.9091
Epoch 84/100
```

```
1/1 [==============================] - 0s 118ms/step - loss: 0.2496 -
f1_score: 0.9524 - val_loss: 0.2153 - val_f1_score: 0.9091
Epoch 85/100
1/1 [==============================] - 0s 175ms/step - loss: 0.2164 -
f1_score: 0.9500 - val_loss: 0.2183 - val_f1_score: 0.9091
Epoch 86/100
1/1 [==============================] - 0s 115ms/step - loss: 0.0344 -
f1_score: 1.0000 - val_loss: 0.2249 - val_f1_score: 0.9091
Epoch 87/100
1/1 [==============================] - 0s 110ms/step - loss: 0.0865 -
f1_score: 0.9524 - val_loss: 0.2325 - val_f1_score: 0.9091
Epoch 88/100
1/1 [==============================] - 0s 117ms/step - loss: 0.0629 -
f1_score: 0.9524 - val_loss: 0.2404 - val_f1_score: 0.9091
Epoch 89/100
1/1 [==============================] - 0s 126ms/step - loss: 0.0339 -
f1_score: 1.0000 - val_loss: 0.2531 - val_f1_score: 0.9091
Epoch 90/100
1/1 [==============================] - 0s 135ms/step - loss: 0.0851 -
f1_score: 0.9302 - val_loss: 0.2627 - val_f1_score: 0.9091
Epoch 91/100
1/1 [==============================] - 0s 130ms/step - loss: 0.0081 -
f1_score: 1.0000 - val_loss: 0.2714 - val_f1_score: 0.9091
Epoch 92/100
1/1 [==============================] - 0s 123ms/step - loss: 0.0438 -
f1_score: 0.9744 - val_loss: 0.2845 - val_f1_score: 0.9091
Epoch 93/100
1/1 [==============================] - 0s 167ms/step - loss: 0.1215 -
f1_score: 0.9756 - val_loss: 0.2967 - val_f1_score: 0.9091
Epoch 94/100
1/1 [==============================] - 0s 131ms/step - loss: 0.0676 -
f1_score: 0.9756 - val_loss: 0.3076 - val_f1_score: 0.9091
Epoch 95/100
1/1 [==============================] - 0s 164ms/step - loss: 0.0019 -
f1_score: 1.0000 - val_loss: 0.3175 - val_f1_score: 0.9091
Epoch 96/100
1/1 [==============================] - 0s 88ms/step - loss: 0.0024 -
f1_score: 1.0000 - val_loss: 0.3266 - val_f1_score: 0.9091
Epoch 97/100
1/1 [==============================] - 0s 102ms/step - loss: 0.0426 -
f1_score: 0.9756 - val_loss: 0.3342 - val_f1_score: 0.9091
Epoch 98/100
1/1 [==============================] - 0s 169ms/step - loss: 0.0349 -
f1_score: 0.9756 - val_loss: 0.3395 - val_f1_score: 0.9091
Epoch 99/100
1/1 [==============================] - 0s 104ms/step - loss: 0.1804 -
f1_score: 0.9744 - val_loss: 0.3595 - val_f1_score: 0.9091
Epoch 100/100
```

```
1/1 [==============================] - 0s 185ms/step - loss: 0.1495 -
f1_score: 0.9744 - val_loss: 0.3744 - val_f1_score: 0.9091
```



Upsampled Dataset

```
train_and_plot_nn(X_u, y_u)
```

```
Epoch 1/100
2/2 [==============================] - 3s 680ms/step - loss: 3.0772 -
f1_score: 0.4478 - val_loss: 0.1581 - val_f1_score: 0.8571
Epoch 2/100
2/2 [==============================] - 0s 158ms/step - loss: 1.4139 -
f1_score: 0.7949 - val_loss: 0.0704 - val_f1_score: 1.0000
Epoch 3/100
2/2 [==============================] - 0s 109ms/step - loss: 2.0004 -
f1_score: 0.8000 - val_loss: 0.0485 - val_f1_score: 1.0000
Epoch 4/100
2/2 [==============================] - 0s 91ms/step - loss: 1.2941 -
f1_score: 0.8395 - val_loss: 0.0437 - val_f1_score: 1.0000
Epoch 5/100
2/2 [==============================] - 0s 106ms/step - loss: 1.1954 -
f1_score: 0.8571 - val_loss: 0.0334 - val_f1_score: 1.0000
Epoch 6/100
2/2 [==============================] - 0s 111ms/step - loss: 1.1770 -
f1_score: 0.8312 - val_loss: 0.0279 - val_f1_score: 1.0000
Epoch 7/100
2/2 [==============================] - 0s 101ms/step - loss: 1.1182 -
f1_score: 0.8500 - val_loss: 0.0225 - val_f1_score: 1.0000
Epoch 8/100
2/2 [==============================] - 0s 109ms/step - loss: 1.1434 -
f1_score: 0.8831 - val_loss: 0.0167 - val_f1_score: 1.0000
Epoch 9/100
2/2 [==============================] - 0s 85ms/step - loss: 0.6602 -
f1_score: 0.9398 - val_loss: 0.0136 - val_f1_score: 1.0000
Epoch 10/100
2/2 [==============================] - 0s 81ms/step - loss: 0.8442 -
f1_score: 0.8642 - val_loss: 0.0134 - val_f1_score: 1.0000
```

```
Epoch 11/100
2/2 [==============================] - 0s 100ms/step - loss: 0.7617 -
f1_score: 0.9250 - val_loss: 0.0151 - val_f1_score: 1.0000
Epoch 12/100
2/2 [==============================] - 0s 146ms/step - loss: 0.4699 -
f1_score: 0.9383 - val_loss: 0.0170 - val_f1_score: 1.0000
Epoch 13/100
2/2 [==============================] - 0s 85ms/step - loss: 1.0169 -
f1_score: 0.9268 - val_loss: 0.0221 - val_f1_score: 1.0000
Epoch 14/100
2/2 [==============================] - 0s 97ms/step - loss: 0.4641 -
f1_score: 0.9136 - val_loss: 0.0246 - val_f1_score: 1.0000
Epoch 15/100
2/2 [==============================] - 0s 92ms/step - loss: 0.4504 -
f1_score: 0.9367 - val_loss: 0.0309 - val_f1_score: 1.0000
Epoch 16/100
2/2 [==============================] - 0s 164ms/step - loss: 0.2866 -
f1_score: 0.9512 - val_loss: 0.0374 - val_f1_score: 0.9333
Epoch 17/100
2/2 [==============================] - 0s 92ms/step - loss: 0.5589 -
f1_score: 0.9211 - val_loss: 0.0439 - val_f1_score: 0.9333
Epoch 18/100
2/2 [==============================] - 0s 61ms/step - loss: 0.4003 -
f1_score: 0.9500 - val_loss: 0.0485 - val_f1_score: 0.9333
Epoch 19/100
2/2 [==============================] - 0s 74ms/step - loss: 1.0340 -
f1_score: 0.9286 - val_loss: 0.0538 - val_f1_score: 0.9333
Epoch 20/100
2/2 [==============================] - 0s 73ms/step - loss: 0.9349 -
f1_score: 0.8916 - val_loss: 0.0594 - val_f1_score: 0.9333
Epoch 21/100
2/2 [==============================] - 0s 80ms/step - loss: 0.1449 -
f1_score: 0.9873 - val_loss: 0.0628 - val_f1_score: 0.9333
Epoch 22/100
2/2 [==============================] - 0s 67ms/step - loss: 0.3431 -
f1_score: 0.9412 - val_loss: 0.0612 - val_f1_score: 0.9333
Epoch 23/100
2/2 [==============================] - 0s 53ms/step - loss: 0.1883 -
f1_score: 0.9500 - val_loss: 0.0608 - val_f1_score: 0.9333
Epoch 24/100
2/2 [==============================] - 0s 40ms/step - loss: 0.0660 -
f1_score: 0.9873 - val_loss: 0.0599 - val_f1_score: 0.9333
Epoch 25/100
2/2 [==============================] - 0s 41ms/step - loss: 0.2754 -
f1_score: 0.9750 - val_loss: 0.0584 - val_f1_score: 0.9333
Epoch 26/100
2/2 [==============================] - 0s 43ms/step - loss: 0.0391 -
f1_score: 0.9877 - val_loss: 0.0563 - val_f1_score: 0.9333
Epoch 27/100
```

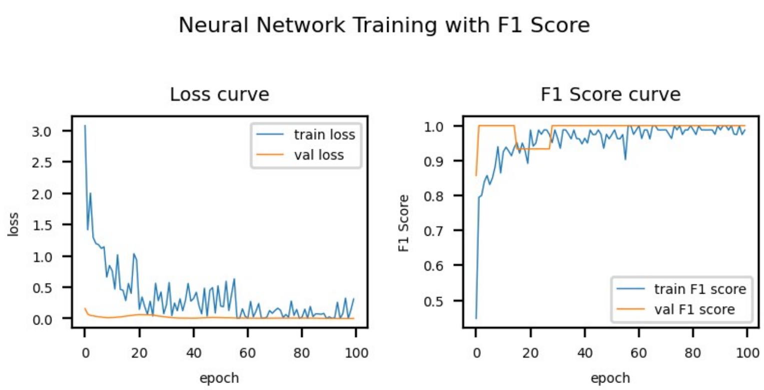
```
2/2 [==============================] - 0s 43ms/step - loss: 0.5623 -
f1_score: 0.9873 - val_loss: 0.0485 - val_f1_score: 0.9333
Epoch 28/100
2/2 [==============================] - 0s 57ms/step - loss: 0.2835 -
f1_score: 0.9744 - val_loss: 0.0399 - val_f1_score: 0.9333
Epoch 29/100
2/2 [==============================] - 0s 41ms/step - loss: 0.4229 -
f1_score: 0.9512 - val_loss: 0.0342 - val_f1_score: 1.0000
Epoch 30/100
2/2 [==============================] - 0s 42ms/step - loss: 0.0763 -
f1_score: 0.9877 - val_loss: 0.0294 - val_f1_score: 1.0000
Epoch 31/100
2/2 [==============================] - 0s 49ms/step - loss: 0.2195 -
f1_score: 0.9639 - val_loss: 0.0237 - val_f1_score: 1.0000
Epoch 32/100
2/2 [==============================] - 0s 39ms/step - loss: 0.5734 -
f1_score: 0.9351 - val_loss: 0.0182 - val_f1_score: 1.0000
Epoch 33/100
2/2 [==============================] - 0s 40ms/step - loss: 0.0465 -
f1_score: 0.9877 - val_loss: 0.0145 - val_f1_score: 1.0000
Epoch 34/100
2/2 [==============================] - 0s 41ms/step - loss: 0.2463 -
f1_score: 0.9877 - val_loss: 0.0111 - val_f1_score: 1.0000
Epoch 35/100
2/2 [==============================] - 0s 42ms/step - loss: 0.1232 -
f1_score: 0.9756 - val_loss: 0.0081 - val_f1_score: 1.0000
Epoch 36/100
2/2 [==============================] - 0s 39ms/step - loss: 0.3148 -
f1_score: 0.9620 - val_loss: 0.0071 - val_f1_score: 1.0000
Epoch 37/100
2/2 [==============================] - 0s 56ms/step - loss: 0.1233 -
f1_score: 0.9873 - val_loss: 0.0063 - val_f1_score: 1.0000
Epoch 38/100
2/2 [==============================] - 0s 55ms/step - loss: 0.2871 -
f1_score: 0.9630 - val_loss: 0.0059 - val_f1_score: 1.0000
Epoch 39/100
2/2 [==============================] - 0s 57ms/step - loss: 0.5592 -
f1_score: 0.9620 - val_loss: 0.0057 - val_f1_score: 1.0000
Epoch 40/100
2/2 [==============================] - 0s 57ms/step - loss: 0.2760 -
f1_score: 0.9474 - val_loss: 0.0051 - val_f1_score: 1.0000
Epoch 41/100
2/2 [==============================] - 0s 41ms/step - loss: 0.3190 -
f1_score: 0.9639 - val_loss: 0.0045 - val_f1_score: 1.0000
Epoch 42/100
2/2 [==============================] - 0s 56ms/step - loss: 0.4190 -
f1_score: 0.9500 - val_loss: 0.0053 - val_f1_score: 1.0000
Epoch 43/100
2/2 [==============================] - 0s 43ms/step - loss: 0.2032 -
```

```
f1_score: 0.9873 - val_loss: 0.0073 - val_f1_score: 1.0000
Epoch 44/100
2/2 [==============================] - 0s 40ms/step - loss: 0.0735 -
f1_score: 0.9744 - val_loss: 0.0096 - val_f1_score: 1.0000
Epoch 45/100
2/2 [==============================] - 0s 44ms/step - loss: 0.4840 -
f1_score: 0.9744 - val_loss: 0.0113 - val_f1_score: 1.0000
Epoch 46/100
2/2 [==============================] - 0s 44ms/step - loss: 0.0372 -
f1_score: 0.9877 - val_loss: 0.0129 - val_f1_score: 1.0000
Epoch 47/100
2/2 [==============================] - 0s 55ms/step - loss: 0.4502 -
f1_score: 0.9750 - val_loss: 0.0148 - val_f1_score: 1.0000
Epoch 48/100
2/2 [==============================] - 0s 56ms/step - loss: 0.4927 -
f1_score: 0.9351 - val_loss: 0.0157 - val_f1_score: 1.0000
Epoch 49/100
2/2 [==============================] - 0s 55ms/step - loss: 0.0870 -
f1_score: 0.9750 - val_loss: 0.0155 - val_f1_score: 1.0000
Epoch 50/100
2/2 [==============================] - 0s 56ms/step - loss: 0.5208 -
f1_score: 0.9620 - val_loss: 0.0146 - val_f1_score: 1.0000
Epoch 51/100
2/2 [==============================] - 0s 54ms/step - loss: 0.1971 -
f1_score: 0.9744 - val_loss: 0.0137 - val_f1_score: 1.0000
Epoch 52/100
2/2 [==============================] - 0s 39ms/step - loss: 0.1884 -
f1_score: 0.9877 - val_loss: 0.0128 - val_f1_score: 1.0000
Epoch 53/100
2/2 [==============================] - 0s 43ms/step - loss: 0.5939 -
f1_score: 0.9630 - val_loss: 0.0116 - val_f1_score: 1.0000
Epoch 54/100
2/2 [==============================] - 0s 64ms/step - loss: 0.1274 -
f1_score: 0.9620 - val_loss: 0.0109 - val_f1_score: 1.0000
Epoch 55/100
2/2 [==============================] - 0s 57ms/step - loss: 0.3849 -
f1_score: 0.9744 - val_loss: 0.0102 - val_f1_score: 1.0000
Epoch 56/100
2/2 [==============================] - 0s 114ms/step - loss: 0.6336 -
f1_score: 0.9024 - val_loss: 0.0089 - val_f1_score: 1.0000
Epoch 57/100
2/2 [==============================] - 0s 77ms/step - loss: 0.0037 -
f1_score: 1.0000 - val_loss: 0.0082 - val_f1_score: 1.0000
Epoch 58/100
2/2 [==============================] - 0s 115ms/step - loss: 0.0058 -
f1_score: 1.0000 - val_loss: 0.0079 - val_f1_score: 1.0000
Epoch 59/100
2/2 [==============================] - 0s 73ms/step - loss: 0.1545 -
f1_score: 0.9750 - val_loss: 0.0076 - val_f1_score: 1.0000
```

```
Epoch 60/100
2/2 [==============================] - 0s 134ms/step - loss: 0.0365 -
f1_score: 0.9873 - val_loss: 0.0070 - val_f1_score: 1.0000
Epoch 61/100
2/2 [==============================] - 0s 84ms/step - loss: 0.0027 -
f1_score: 1.0000 - val_loss: 0.0065 - val_f1_score: 1.0000
Epoch 62/100
2/2 [==============================] - 0s 85ms/step - loss: 0.2725 -
f1_score: 0.9630 - val_loss: 0.0057 - val_f1_score: 1.0000
Epoch 63/100
2/2 [==============================] - 0s 91ms/step - loss: 0.0257 -
f1_score: 0.9877 - val_loss: 0.0052 - val_f1_score: 1.0000
Epoch 64/100
2/2 [==============================] - 0s 68ms/step - loss: 0.1163 -
f1_score: 0.9873 - val_loss: 0.0047 - val_f1_score: 1.0000
Epoch 65/100
2/2 [==============================] - 0s 78ms/step - loss: 0.2380 -
f1_score: 0.9620 - val_loss: 0.0046 - val_f1_score: 1.0000
Epoch 66/100
2/2 [==============================] - 0s 97ms/step - loss: 0.0017 -
f1_score: 1.0000 - val_loss: 0.0046 - val_f1_score: 1.0000
Epoch 67/100
2/2 [==============================] - 0s 77ms/step - loss: 0.0063 -
f1_score: 1.0000 - val_loss: 0.0047 - val_f1_score: 1.0000
Epoch 68/100
2/2 [==============================] - 0s 213ms/step - loss: 0.0211 -
f1_score: 0.9877 - val_loss: 0.0047 - val_f1_score: 1.0000
Epoch 69/100
2/2 [==============================] - 0s 157ms/step - loss: 0.1270 -
f1_score: 0.9873 - val_loss: 0.0050 - val_f1_score: 1.0000
Epoch 70/100
2/2 [==============================] - 0s 203ms/step - loss: 0.0834 -
f1_score: 0.9877 - val_loss: 0.0056 - val_f1_score: 1.0000
Epoch 71/100
2/2 [==============================] - 0s 148ms/step - loss: 0.1271 -
f1_score: 0.9877 - val_loss: 0.0055 - val_f1_score: 1.0000
Epoch 72/100
2/2 [==============================] - 0s 149ms/step - loss: 0.1639 -
f1_score: 0.9756 - val_loss: 0.0058 - val_f1_score: 1.0000
Epoch 73/100
2/2 [==============================] - 0s 93ms/step - loss: 0.1330 -
f1_score: 0.9630 - val_loss: 0.0058 - val_f1_score: 1.0000
Epoch 74/100
2/2 [==============================] - 0s 73ms/step - loss: 0.0150 -
f1_score: 1.0000 - val_loss: 0.0061 - val_f1_score: 1.0000
Epoch 75/100
2/2 [==============================] - 0s 105ms/step - loss: 0.0644 -
f1_score: 0.9877 - val_loss: 0.0062 - val_f1_score: 1.0000
Epoch 76/100
```

```
2/2 [==============================] - 0s 74ms/step - loss: 0.0191 -
f1_score: 1.0000 - val_loss: 0.0063 - val_f1_score: 1.0000
Epoch 77/100
2/2 [==============================] - 0s 76ms/step - loss: 0.2782 -
f1_score: 0.9744 - val_loss: 0.0064 - val_f1_score: 1.0000
Epoch 78/100
2/2 [==============================] - 0s 75ms/step - loss: 0.0497 -
f1_score: 0.9877 - val_loss: 0.0064 - val_f1_score: 1.0000
Epoch 79/100
2/2 [==============================] - 0s 64ms/step - loss: 0.1444 -
f1_score: 0.9873 - val_loss: 0.0065 - val_f1_score: 1.0000
Epoch 80/100
2/2 [==============================] - 0s 115ms/step - loss: 9.2414e-
04 - f1_score: 1.0000 - val_loss: 0.0067 - val_f1_score: 1.0000
Epoch 81/100
2/2 [==============================] - 0s 61ms/step - loss: 0.0248 -
f1_score: 0.9877 - val_loss: 0.0067 - val_f1_score: 1.0000
Epoch 82/100
2/2 [==============================] - 0s 65ms/step - loss: 0.1488 -
f1_score: 0.9744 - val_loss: 0.0067 - val_f1_score: 1.0000
Epoch 83/100
2/2 [==============================] - 0s 66ms/step - loss: 9.0058e-04
- f1_score: 1.0000 - val_loss: 0.0065 - val_f1_score: 1.0000
Epoch 84/100
2/2 [==============================] - 0s 64ms/step - loss: 0.1925 -
f1_score: 0.9877 - val_loss: 0.0060 - val_f1_score: 1.0000
Epoch 85/100
2/2 [==============================] - 0s 88ms/step - loss: 0.0303 -
f1_score: 0.9877 - val_loss: 0.0055 - val_f1_score: 1.0000
Epoch 86/100
2/2 [==============================] - 0s 212ms/step - loss: 0.0749 -
f1_score: 0.9873 - val_loss: 0.0049 - val_f1_score: 1.0000
Epoch 87/100
2/2 [==============================] - 0s 118ms/step - loss: 0.0735 -
f1_score: 0.9877 - val_loss: 0.0039 - val_f1_score: 1.0000
Epoch 88/100
2/2 [==============================] - 0s 79ms/step - loss: 0.0672 -
f1_score: 0.9877 - val_loss: 0.0028 - val_f1_score: 1.0000
Epoch 89/100
2/2 [==============================] - 0s 61ms/step - loss: 0.0817 -
f1_score: 0.9750 - val_loss: 0.0021 - val_f1_score: 1.0000
Epoch 90/100
2/2 [==============================] - 0s 85ms/step - loss: 3.1672e-05
- f1_score: 1.0000 - val_loss: 0.0017 - val_f1_score: 1.0000
Epoch 91/100
2/2 [==============================] - 0s 177ms/step - loss: 0.0264 -
f1_score: 0.9877 - val_loss: 0.0014 - val_f1_score: 1.0000
Epoch 92/100
2/2 [==============================] - 0s 79ms/step - loss: 0.0034 -
```

```
f1_score: 1.0000 - val_loss: 0.0012 - val_f1_score: 1.0000
Epoch 93/100
2/2 [==============================] - 0s 125ms/step - loss: 0.0096 -
f1_score: 1.0000 - val_loss: 9.6488e-04 - val_f1_score: 1.0000
Epoch 94/100
2/2 [==============================] - 0s 166ms/step - loss: 0.2588 -
f1_score: 0.9873 - val_loss: 9.0126e-04 - val_f1_score: 1.0000
Epoch 95/100
2/2 [==============================] - 0s 74ms/step - loss: 5.8866e-05
- f1_score: 1.0000 - val_loss: 9.2295e-04 - val_f1_score: 1.0000
Epoch 96/100
2/2 [==============================] - 0s 102ms/step - loss: 0.0796 -
f1_score: 0.9756 - val_loss: 9.4623e-04 - val_f1_score: 1.0000
Epoch 97/100
2/2 [==============================] - 0s 141ms/step - loss: 0.3254 -
f1_score: 0.9744 - val_loss: 9.0632e-04 - val_f1_score: 1.0000
Epoch 98/100
2/2 [==============================] - 0s 88ms/step - loss: 1.7195e-04
- f1_score: 1.0000 - val_loss: 8.7483e-04 - val_f1_score: 1.0000
Epoch 99/100
2/2 [==============================] - 0s 166ms/step - loss: 0.1397 -
f1_score: 0.9744 - val_loss: 8.3308e-04 - val_f1_score: 1.0000
Epoch 100/100
2/2 [==============================] - 0s 178ms/step - loss: 0.3115 -
f1_score: 0.9873 - val_loss: 8.2670e-04 - val_f1_score: 1.0000
```



Downsampled Dataset

```
train_and_plot_nn(X_d, y_d)

Epoch 1/100
1/1 [==============================] - 1s 1s/step - loss: 2.1465 -
f1_score: 0.5581 - val_loss: 0.5910 - val_f1_score: 0.8000
Epoch 2/100
1/1 [==============================] - 0s 72ms/step - loss: 3.1036 -
f1_score: 0.5789 - val_loss: 0.2976 - val_f1_score: 0.8571
Epoch 3/100
```

```
1/1 [==============================] - 0s 61ms/step - loss: 1.5248 -
f1_score: 0.7692 - val_loss: 0.1886 - val_f1_score: 0.9333
Epoch 4/100
1/1 [==============================] - 0s 60ms/step - loss: 0.7379 -
f1_score: 0.9444 - val_loss: 0.2008 - val_f1_score: 0.8750
Epoch 5/100
1/1 [==============================] - 0s 46ms/step - loss: 2.0183 -
f1_score: 0.7568 - val_loss: 0.2273 - val_f1_score: 0.8750
Epoch 6/100
1/1 [==============================] - 0s 61ms/step - loss: 1.5012 -
f1_score: 0.6857 - val_loss: 0.2442 - val_f1_score: 0.8750
Epoch 7/100
1/1 [==============================] - 0s 46ms/step - loss: 1.1242 -
f1_score: 0.8000 - val_loss: 0.2760 - val_f1_score: 0.8750
Epoch 8/100
1/1 [==============================] - 0s 51ms/step - loss: 1.7748 -
f1_score: 0.8000 - val_loss: 0.2532 - val_f1_score: 0.8750
Epoch 9/100
1/1 [==============================] - 0s 45ms/step - loss: 1.2814 -
f1_score: 0.8485 - val_loss: 0.1841 - val_f1_score: 0.8750
Epoch 10/100
1/1 [==============================] - 0s 46ms/step - loss: 0.2409 -
f1_score: 0.9730 - val_loss: 0.1582 - val_f1_score: 0.9333
Epoch 11/100
1/1 [==============================] - 0s 49ms/step - loss: 1.5642 -
f1_score: 0.8889 - val_loss: 0.1723 - val_f1_score: 0.9333
Epoch 12/100
1/1 [==============================] - 0s 52ms/step - loss: 0.7556 -
f1_score: 0.8824 - val_loss: 0.1755 - val_f1_score: 0.9333
Epoch 13/100
1/1 [==============================] - 0s 59ms/step - loss: 0.4863 -
f1_score: 0.8649 - val_loss: 0.1545 - val_f1_score: 0.9333
Epoch 14/100
1/1 [==============================] - 0s 61ms/step - loss: 0.4838 -
f1_score: 0.8947 - val_loss: 0.1369 - val_f1_score: 0.9333
Epoch 15/100
1/1 [==============================] - 0s 64ms/step - loss: 0.0321 -
f1_score: 1.0000 - val_loss: 0.1285 - val_f1_score: 0.9333
Epoch 16/100
1/1 [==============================] - 0s 48ms/step - loss: 0.2022 -
f1_score: 0.9189 - val_loss: 0.1317 - val_f1_score: 0.9333
Epoch 17/100
1/1 [==============================] - 0s 45ms/step - loss: 0.1951 -
f1_score: 0.9412 - val_loss: 0.1357 - val_f1_score: 0.9333
Epoch 18/100
1/1 [==============================] - 0s 47ms/step - loss: 0.8839 -
f1_score: 0.9143 - val_loss: 0.1447 - val_f1_score: 0.9333
Epoch 19/100
1/1 [==============================] - 0s 60ms/step - loss: 1.6183 -
```

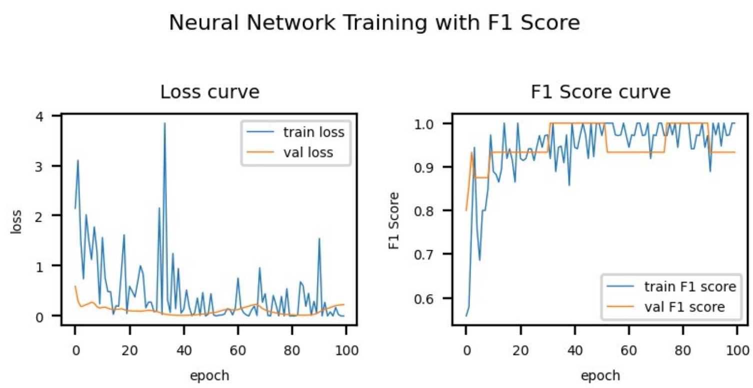
```
f1_score: 0.8649 - val_loss: 0.1274 - val_f1_score: 0.9333
Epoch 20/100
1/1 [==============================] - 0s 46ms/step - loss: 0.0480 -
f1_score: 1.0000 - val_loss: 0.1111 - val_f1_score: 0.9333
Epoch 21/100
1/1 [==============================] - 0s 57ms/step - loss: 0.5911 -
f1_score: 0.9189 - val_loss: 0.1038 - val_f1_score: 0.9333
Epoch 22/100
1/1 [==============================] - 0s 61ms/step - loss: 0.4889 -
f1_score: 0.9143 - val_loss: 0.0985 - val_f1_score: 0.9333
Epoch 23/100
1/1 [==============================] - 0s 61ms/step - loss: 0.3771 -
f1_score: 0.9189 - val_loss: 0.0975 - val_f1_score: 0.9333
Epoch 24/100
1/1 [==============================] - 0s 47ms/step - loss: 0.6764 -
f1_score: 0.9412 - val_loss: 0.0995 - val_f1_score: 0.9333
Epoch 25/100
1/1 [==============================] - 0s 65ms/step - loss: 1.0033 -
f1_score: 0.9412 - val_loss: 0.0937 - val_f1_score: 0.9333
Epoch 26/100
1/1 [==============================] - 0s 61ms/step - loss: 0.8366 -
f1_score: 0.9143 - val_loss: 0.0993 - val_f1_score: 0.9333
Epoch 27/100
1/1 [==============================] - 0s 62ms/step - loss: 0.1618 -
f1_score: 0.9444 - val_loss: 0.1082 - val_f1_score: 0.9333
Epoch 28/100
1/1 [==============================] - 0s 64ms/step - loss: 0.2768 -
f1_score: 0.9714 - val_loss: 0.1149 - val_f1_score: 0.9333
Epoch 29/100
1/1 [==============================] - 0s 62ms/step - loss: 0.2732 -
f1_score: 0.9444 - val_loss: 0.1095 - val_f1_score: 0.9333
Epoch 30/100
1/1 [==============================] - 0s 51ms/step - loss: 0.0847 -
f1_score: 0.9714 - val_loss: 0.1024 - val_f1_score: 0.9333
Epoch 31/100
1/1 [==============================] - 0s 64ms/step - loss: 0.0978 -
f1_score: 0.9730 - val_loss: 0.0920 - val_f1_score: 0.9333
Epoch 32/100
1/1 [==============================] - 0s 45ms/step - loss: 2.1537 -
f1_score: 0.9189 - val_loss: 0.0713 - val_f1_score: 1.0000
Epoch 33/100
1/1 [==============================] - 0s 62ms/step - loss: 0.0275 -
f1_score: 1.0000 - val_loss: 0.0552 - val_f1_score: 1.0000
Epoch 34/100
1/1 [==============================] - 0s 62ms/step - loss: 3.8435 -
f1_score: 0.8889 - val_loss: 0.0368 - val_f1_score: 1.0000
Epoch 35/100
1/1 [==============================] - 0s 66ms/step - loss: 0.3270 -
f1_score: 0.9444 - val_loss: 0.0283 - val_f1_score: 1.0000
```

```
Epoch 36/100
1/1 [==============================] - 0s 45ms/step - loss: 0.0708 -
f1_score: 0.9474 - val_loss: 0.0223 - val_f1_score: 1.0000
Epoch 37/100
1/1 [==============================] - 0s 53ms/step - loss: 1.2459 -
f1_score: 0.9091 - val_loss: 0.0182 - val_f1_score: 1.0000
Epoch 38/100
1/1 [==============================] - 0s 45ms/step - loss: 0.1447 -
f1_score: 0.9730 - val_loss: 0.0139 - val_f1_score: 1.0000
Epoch 39/100
1/1 [==============================] - 0s 45ms/step - loss: 0.9470 -
f1_score: 0.8571 - val_loss: 0.0135 - val_f1_score: 1.0000
Epoch 40/100
1/1 [==============================] - 0s 46ms/step - loss: 0.0571 -
f1_score: 1.0000 - val_loss: 0.0130 - val_f1_score: 1.0000
Epoch 41/100
1/1 [==============================] - 0s 47ms/step - loss: 0.1347 -
f1_score: 0.9444 - val_loss: 0.0127 - val_f1_score: 1.0000
Epoch 42/100
1/1 [==============================] - 0s 47ms/step - loss: 0.5188 -
f1_score: 0.9412 - val_loss: 0.0134 - val_f1_score: 1.0000
Epoch 43/100
1/1 [==============================] - 0s 61ms/step - loss: 0.1808 -
f1_score: 0.9714 - val_loss: 0.0155 - val_f1_score: 1.0000
Epoch 44/100
1/1 [==============================] - 0s 59ms/step - loss: 0.0063 -
f1_score: 1.0000 - val_loss: 0.0178 - val_f1_score: 1.0000
Epoch 45/100
1/1 [==============================] - 0s 62ms/step - loss: 0.0599 -
f1_score: 0.9730 - val_loss: 0.0200 - val_f1_score: 1.0000
Epoch 46/100
1/1 [==============================] - 0s 62ms/step - loss: 0.3590 -
f1_score: 0.9189 - val_loss: 0.0258 - val_f1_score: 1.0000
Epoch 47/100
1/1 [==============================] - 0s 48ms/step - loss: 0.0107 -
f1_score: 1.0000 - val_loss: 0.0330 - val_f1_score: 1.0000
Epoch 48/100
1/1 [==============================] - 0s 50ms/step - loss: 0.4669 -
f1_score: 0.9231 - val_loss: 0.0382 - val_f1_score: 1.0000
Epoch 49/100
1/1 [==============================] - 0s 65ms/step - loss: 0.0011 -
f1_score: 1.0000 - val_loss: 0.0439 - val_f1_score: 1.0000
Epoch 50/100
1/1 [==============================] - 0s 59ms/step - loss: 0.0326 -
f1_score: 1.0000 - val_loss: 0.0522 - val_f1_score: 1.0000
Epoch 51/100
1/1 [==============================] - 0s 44ms/step - loss: 0.4439 -
f1_score: 0.9714 - val_loss: 0.0610 - val_f1_score: 1.0000
Epoch 52/100
```

```
1/1 [==============================] - 0s 62ms/step - loss: 0.0200 -
f1_score: 1.0000 - val_loss: 0.0680 - val_f1_score: 1.0000
Epoch 53/100
1/1 [==============================] - 0s 44ms/step - loss: 4.7986e-04
- f1_score: 1.0000 - val_loss: 0.0748 - val_f1_score: 0.9333
Epoch 54/100
1/1 [==============================] - 0s 60ms/step - loss: 0.0163 -
f1_score: 1.0000 - val_loss: 0.0918 - val_f1_score: 0.9333
Epoch 55/100
1/1 [==============================] - 0s 65ms/step - loss: 0.0181 -
f1_score: 1.0000 - val_loss: 0.1090 - val_f1_score: 0.9333
Epoch 56/100
1/1 [==============================] - 0s 46ms/step - loss: 0.0376 -
f1_score: 0.9730 - val_loss: 0.1242 - val_f1_score: 0.9333
Epoch 57/100
1/1 [==============================] - 0s 47ms/step - loss: 0.1429 -
f1_score: 0.9714 - val_loss: 0.1284 - val_f1_score: 0.9333
Epoch 58/100
1/1 [==============================] - 0s 63ms/step - loss: 0.1329 -
f1_score: 0.9730 - val_loss: 0.1249 - val_f1_score: 0.9333
Epoch 59/100
1/1 [==============================] - 0s 45ms/step - loss: 0.0206 -
f1_score: 1.0000 - val_loss: 0.1224 - val_f1_score: 0.9333
Epoch 60/100
1/1 [==============================] - 0s 50ms/step - loss: 0.1586 -
f1_score: 0.9714 - val_loss: 0.1217 - val_f1_score: 0.9333
Epoch 61/100
1/1 [==============================] - 0s 45ms/step - loss: 0.7558 -
f1_score: 0.9444 - val_loss: 0.1270 - val_f1_score: 0.9333
Epoch 62/100
1/1 [==============================] - 0s 47ms/step - loss: 0.1763 -
f1_score: 0.9730 - val_loss: 0.1478 - val_f1_score: 0.9333
Epoch 63/100
1/1 [==============================] - 0s 46ms/step - loss: 0.0694 -
f1_score: 0.9714 - val_loss: 0.1650 - val_f1_score: 0.9333
Epoch 64/100
1/1 [==============================] - 0s 47ms/step - loss: 0.0217 -
f1_score: 1.0000 - val_loss: 0.1794 - val_f1_score: 0.9333
Epoch 65/100
1/1 [==============================] - 0s 71ms/step - loss: 2.6224e-04
- f1_score: 1.0000 - val_loss: 0.1964 - val_f1_score: 0.9333
Epoch 66/100
1/1 [==============================] - 0s 64ms/step - loss: 0.1197 -
f1_score: 0.9714 - val_loss: 0.2132 - val_f1_score: 0.9333
Epoch 67/100
1/1 [==============================] - 0s 46ms/step - loss: 0.1958 -
f1_score: 0.9730 - val_loss: 0.2253 - val_f1_score: 0.9333
Epoch 68/100
1/1 [==============================] - 0s 62ms/step - loss: 0.0091 -
```

```
f1_score: 1.0000 - val_loss: 0.2349 - val_f1_score: 0.9333
Epoch 69/100
1/1 [===============================] - 0s 62ms/step - loss: 0.9589 -
f1_score: 0.9189 - val_loss: 0.1917 - val_f1_score: 0.9333
Epoch 70/100
1/1 [===============================] - 0s 61ms/step - loss: 0.2731 -
f1_score: 0.9730 - val_loss: 0.1480 - val_f1_score: 0.9333
Epoch 71/100
1/1 [===============================] - 0s 45ms/step - loss: 0.4447 -
f1_score: 0.9714 - val_loss: 0.1211 - val_f1_score: 0.9333
Epoch 72/100
1/1 [===============================] - 0s 76ms/step - loss: 0.0124 -
f1_score: 1.0000 - val_loss: 0.1018 - val_f1_score: 0.9333
Epoch 73/100
1/1 [===============================] - 0s 60ms/step - loss: 2.6735e-05
- f1_score: 1.0000 - val_loss: 0.0858 - val_f1_score: 0.9333
Epoch 74/100
1/1 [===============================] - 0s 61ms/step - loss: 0.4092 -
f1_score: 0.9714 - val_loss: 0.0728 - val_f1_score: 0.9333
Epoch 75/100
1/1 [===============================] - 0s 54ms/step - loss: 0.2248 -
f1_score: 0.9714 - val_loss: 0.0606 - val_f1_score: 1.0000
Epoch 76/100
1/1 [===============================] - 0s 47ms/step - loss: 8.3507e-04
- f1_score: 1.0000 - val_loss: 0.0467 - val_f1_score: 1.0000
Epoch 77/100
1/1 [===============================] - 0s 65ms/step - loss: 0.3914 -
f1_score: 0.9730 - val_loss: 0.0405 - val_f1_score: 1.0000
Epoch 78/100
1/1 [===============================] - 0s 82ms/step - loss: 0.0012 -
f1_score: 1.0000 - val_loss: 0.0359 - val_f1_score: 1.0000
Epoch 79/100
1/1 [===============================] - 0s 82ms/step - loss: 0.5447 -
f1_score: 0.9444 - val_loss: 0.0300 - val_f1_score: 1.0000
Epoch 80/100
1/1 [===============================] - 0s 78ms/step - loss: 1.1233e-05
- f1_score: 1.0000 - val_loss: 0.0255 - val_f1_score: 1.0000
Epoch 81/100
1/1 [===============================] - 0s 84ms/step - loss: 0.0013 -
f1_score: 1.0000 - val_loss: 0.0220 - val_f1_score: 1.0000
Epoch 82/100
1/1 [===============================] - 0s 68ms/step - loss: 3.3509e-06
- f1_score: 1.0000 - val_loss: 0.0192 - val_f1_score: 1.0000
Epoch 83/100
1/1 [===============================] - 0s 61ms/step - loss: 0.0246 -
f1_score: 1.0000 - val_loss: 0.0168 - val_f1_score: 1.0000
Epoch 84/100
1/1 [===============================] - 0s 60ms/step - loss: 0.6814 -
f1_score: 0.9412 - val_loss: 0.0160 - val_f1_score: 1.0000
```

```
Epoch 85/100
1/1 [==============================] - 0s 75ms/step - loss: 0.6012 -
f1_score: 0.9412 - val_loss: 0.0151 - val_f1_score: 1.0000
Epoch 86/100
1/1 [==============================] - 0s 77ms/step - loss: 0.1908 -
f1_score: 0.9730 - val_loss: 0.0162 - val_f1_score: 1.0000
Epoch 87/100
1/1 [==============================] - 0s 68ms/step - loss: 0.4536 -
f1_score: 0.9714 - val_loss: 0.0183 - val_f1_score: 1.0000
Epoch 88/100
1/1 [==============================] - 0s 69ms/step - loss: 0.0108 -
f1_score: 1.0000 - val_loss: 0.0211 - val_f1_score: 1.0000
Epoch 89/100
1/1 [==============================] - 0s 93ms/step - loss: 0.2911 -
f1_score: 0.9444 - val_loss: 0.0347 - val_f1_score: 1.0000
Epoch 90/100
1/1 [==============================] - 0s 62ms/step - loss: 0.0550 -
f1_score: 0.9714 - val_loss: 0.0500 - val_f1_score: 1.0000
Epoch 91/100
1/1 [==============================] - 0s 96ms/step - loss: 1.5456 -
f1_score: 0.8889 - val_loss: 0.0809 - val_f1_score: 0.9333
Epoch 92/100
1/1 [==============================] - 0s 71ms/step - loss: 8.0172e-05
- f1_score: 1.0000 - val_loss: 0.1093 - val_f1_score: 0.9333
Epoch 93/100
1/1 [==============================] - 0s 83ms/step - loss: 0.2727 -
f1_score: 0.9730 - val_loss: 0.1319 - val_f1_score: 0.9333
Epoch 94/100
1/1 [==============================] - 0s 81ms/step - loss: 1.2184e-04
- f1_score: 1.0000 - val_loss: 0.1510 - val_f1_score: 0.9333
Epoch 95/100
1/1 [==============================] - 0s 64ms/step - loss: 0.0809 -
f1_score: 0.9474 - val_loss: 0.1663 - val_f1_score: 0.9333
Epoch 96/100
1/1 [==============================] - 0s 81ms/step - loss: 7.9618e-04
- f1_score: 1.0000 - val_loss: 0.1902 - val_f1_score: 0.9333
Epoch 97/100
1/1 [==============================] - 0s 82ms/step - loss: 0.1708 -
f1_score: 0.9714 - val_loss: 0.2067 - val_f1_score: 0.9333
Epoch 98/100
1/1 [==============================] - 0s 63ms/step - loss: 0.0301 -
f1_score: 0.9730 - val_loss: 0.2159 - val_f1_score: 0.9333
Epoch 99/100
1/1 [==============================] - 0s 83ms/step - loss: 3.2000e-04
- f1_score: 1.0000 - val_loss: 0.2221 - val_f1_score: 0.9333
Epoch 100/100
1/1 [==============================] - 0s 64ms/step - loss: 5.5164e-05
- f1_score: 1.0000 - val_loss: 0.2269 - val_f1_score: 0.9333
```



Justification for Downsampling and Upsampling Strategies in Imbalanced Data Handling

Imbalanced Data Challenges:

F1 Score as Evaluation Metric: The F1 score is chosen as the evaluation metric due to its balanced consideration of precision and recall. It is robust for imbalanced datasets, particularly when the minority class is of greater interest. F1 score offers stability, being less influenced by class imbalance, and allows threshold adjustments for improved model performance. Resampling Strategies: Resampling is a common approach to address class imbalance. Two strategies are employed: downsampling the majority class (ALL) and upsampling using SMOTE.

Downsampling (Downsampled Dataset):

- Objective: Train on a more balanced dataset by downsampling the majority class (ALL).
- Implementation: Randomly select a subset of ALL samples to match the number of AML samples.
- Effect: Improves balance, making the training set moderately imbalanced with a better proportion of positives to negatives.

Upsampling (Upsampled Dataset):

- Objective: Address class imbalance by generating synthetic samples for the minority class (AML) using SMOTE.
- Implementation: Initialize SMOTE and resample the data to achieve a more balanced distribution.
- Effect: Increases the number of minority class samples, contributing to a more balanced dataset.

Neural Network Training and Evaluation:

- Neural Network Architecture: A neural network model is built with a custom F1 score metric. Training is performed on the original, upsampled, and downsampled datasets.
- Class Weights: Class weights are calculated based on the distribution of the target variable.
- Training and Evaluation: The F1 score is monitored during training to assess the model's performance.

Justification for Effectiveness:

1. F1 Score as Evaluation Metric:

- F1 score provides a balanced assessment, crucial for imbalanced datasets.
- Stability in performance evaluation, especially when the minority class is of higher interest.

1. Downsampling:

- Advantages:
  - Improves balance, addressing the challenge of insufficient learning from the minority class.
  - Faster convergence by focusing more on the minority class during training.
  - Efficient disk space utilization by consolidating the majority class into fewer examples.

1. Upsampling:

- Advantages:
  - Generates synthetic samples, enhancing the representation of the minority class.
  - Addresses issues of insufficient positive examples, improving model learning.

1. Neural Network Training:

- Models trained on both upsampled and downsampled datasets demonstrate the adaptability of the neural network to different strategies.
- Evaluation using F1 score allows a comprehensive understanding of model performance, considering both precision and recall.

In conclusion, both downsampling and upsampling strategies are effective for handling imbalanced data, addressing challenges related to biased learning, convergence speed, and resource utilization. The choice between the two depends on the specific characteristics of the dataset and the desired trade-offs in model performance.

Question 3

```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from tensorflow import keras
```

```
from tensorflow.keras import layers
import numpy as np

X = pd.read_csv("gene_exp_X")
X.set_index('Gene Accession Number', inplace=True)
X
```

```
                       AFFX-BioB-5_at  AFFX-BioB-M_at  AFFX-BioB-3_at  \
Gene Accession Number

1                                -214            -153             -58

2                                -139             -73              -1

3                                 -76             -49            -307

4                                -135            -114             265

5                                -106            -125             -76

...                               ...             ...             ...

68                               -154            -136              49

69                                -79            -118             -30

70                                -55             -44              12

71                                -59            -114              23

72                               -131            -126             -50


                       AFFX-BioC-5_at  AFFX-BioC-3_at  AFFX-BioDn-5_at  \
Gene Accession Number

1                                  88            -295             -558

2                                 283            -264             -400

3                                 309            -376             -650

4                                  12            -419             -585

5                                 168            -230             -284

...                               ...             ...              ...

68                                180            -257             -273
```

```
69                                  68           -110          -264

70                                 129           -108          -301

71                                 146           -171          -227

72                                 211           -206          -287


                       AFFX-BioDn-3_at  AFFX-CreX-5_at  AFFX-CreX-3_at
\
Gene Accession Number

1                                  199            -176             252

2                                 -330            -168             101

3                                   33            -367             206

4                                  158            -253              49

5                                    4            -122              70

...                                ...             ...             ...

68                                 141            -123              52

69                                 -28             -61              40

70                                -222            -133             136

71                                 -73            -126              -6

72                                 -34            -114              62


                       AFFX-BioB-5_st  ...  U48730_at  U58516_at
U73738_at  \
Gene Accession Number                  ...

1                                 206  ...        185        511
-125
2                                  74  ...        169        837
-36
3                                -215  ...        315       1199
33
4                                  31  ...        240        835
218
5                                 252  ...        156        649
57
...                               ...  ...        ...        ...
```

```
...
68                                    878  ...          214          540
13
69                                   -217  ...          409          617
-34
70                                    320  ...          131          318
35
71                                    149  ...          214          760
-38
72                                    341  ...          206          697
3

                       X06956_at  X16699_at  X83863_at  Z17240_at  \
Gene Accession Number
1                            389        -37        793        329
2                            442        -17        782        295
3                            168         52       1138        777
4                            174       -110        627        170
5                            504        -26        250        314
...                          ...        ...        ...        ...
68                          1075        -45        524        249
69                           738         11        742        234
70                           241        -66        320        174
71                           201        -55        348        208
72                          1046         27        874        393

                       L49218_f_at  M71243_f_at  Z78285_f_at
Gene Accession Number
1                               36          191          -37
2                               11           76          -14
3                               41          228          -41
4                              -50          126          -91
5                               14           56          -25
...                            ...          ...          ...
68                              40          -68           -1
69                              72          109          -30
70                              -4          176           40
71                               0           74          -12
72                              34          237           -2

[72 rows x 7129 columns]

y = pd.read_csv("gene_exp_y")
y.set_index('patient', inplace=True)
y

         cancer
patient
1             0
2             0
```

```
3              0
4              0
5              0
...          ...
68             0
69             0
70             0
71             0
72             0

[72 rows x 1 columns]
```

Therefore there are 72 sample to train the model, for the given dataset.

Method 1 : K-Fold cross validation

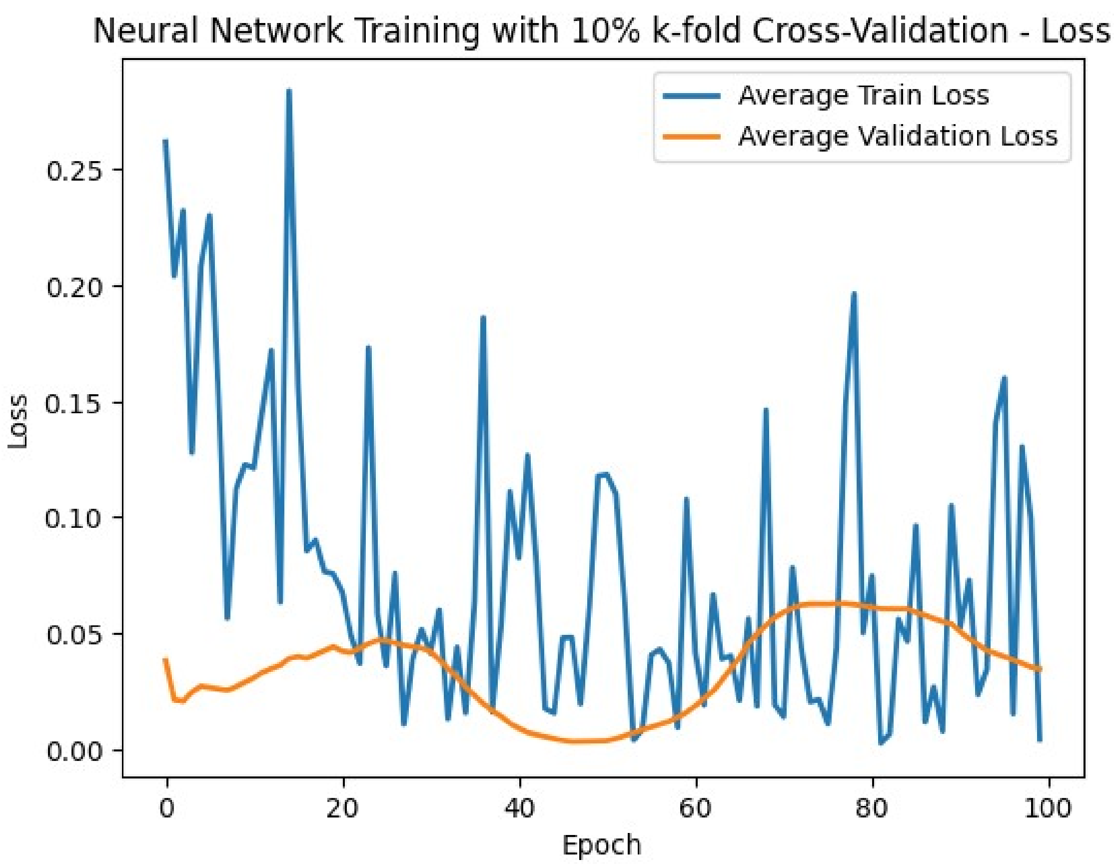
K-Fold validation divides a dataset into k subsets, training the model k times. Each time, one of the k subsets is used as the validation set, and the remaining k-1 subsets form the training set. This helps assess model performance by providing multiple evaluations, reducing the impact of data partitioning.

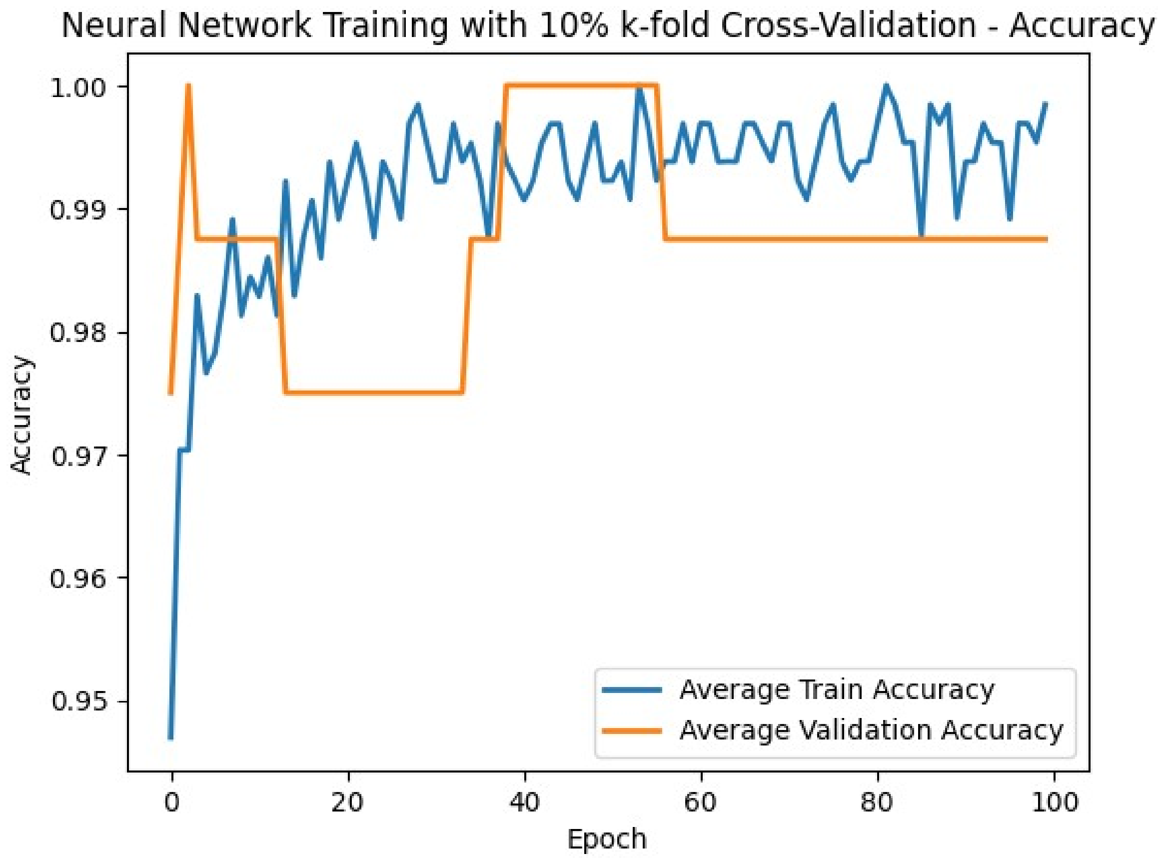
```
from sklearn.model_selection import StratifiedKFold

def nn_kfold(X, y, n_splits=10, random_state=42, epochs=100, 
batch_size=64):
    # Standardize the input features
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    y_array = y.to_numpy()

    class_weights = {0: 1.5405405405405406, 1: 2.85}

    # Build the neural network model
    NN_model = keras.Sequential([
        layers.Dense(64, activation='relu', 
input_shape=X_scaled.shape[1:]),
        layers.Dropout(0.5),
        layers.Dense(32, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(1, activation='sigmoid')
    ])

    NN_model.compile(
        loss='binary_crossentropy',
        optimizer=keras.optimizers.Adam(learning_rate=0.001),
        metrics=['accuracy']
    )
```

```
    # Initialize StratifiedKFold
    skf = StratifiedKFold(n_splits=n_splits,
random_state=random_state, shuffle=True)

    all_train_loss = []
    all_val_loss = []
    all_train_accuracy = []
    all_val_accuracy = []

    # Train and evaluate the model using k-fold cross-validation
    for fold_num, (train_index, val_index) in
enumerate(skf.split(X_scaled, y_array), 1):
        print(f"Training Fold {fold_num}/{n_splits}... for {epochs}
epochs")

        # Split the data into training and validation sets
        X_train, X_val = X_scaled[train_index], X_scaled[val_index]
        y_train, y_val = y_array[train_index], y_array[val_index]

        history = NN_model.fit(
            X_train, y_train,
            validation_data=(X_val, y_val),
class_weight=class_weights,
            batch_size=batch_size,
            epochs=epochs,
            verbose=0
        )

        all_train_loss.append(history.history['loss'])
        all_val_loss.append(history.history['val_loss'])
        all_train_accuracy.append(history.history['accuracy'])
        all_val_accuracy.append(history.history['val_accuracy'])

    plot_average_history(all_train_loss, all_val_loss, "Loss")
    plot_average_history(all_train_accuracy, all_val_accuracy,
"Accuracy")

def plot_average_history(all_train_values, all_val_values,
metric_name):
    avg_train_values = np.mean(all_train_values, axis=0)
    avg_val_values = np.mean(all_val_values, axis=0)

    plt.plot(avg_train_values, label="Average Train " + metric_name,
lw=2)
    plt.plot(avg_val_values, label="Average Validation " +
metric_name, lw=2)

    plt.xlabel("Epoch")
    plt.ylabel(metric_name)
    plt.title(f"Neural Network Training with 10% k-fold Cross-
```

```
Validation - {metric_name}")
    plt.legend()
    plt.show()

nn_kfold(X, y)
```

```
Training Fold 1/10... for 100 epochs
Training Fold 2/10... for 100 epochs
Training Fold 3/10... for 100 epochs
Training Fold 4/10... for 100 epochs
Training Fold 5/10... for 100 epochs
Training Fold 6/10... for 100 epochs
Training Fold 7/10... for 100 epochs
Training Fold 8/10... for 100 epochs
Training Fold 9/10... for 100 epochs
Training Fold 10/10... for 100 epochs
```





Method 2 :Leave-One-Out Cross-Validation

LOOCV, suitable for limited datasets, involves training on all but one sample and validating on the excluded sample iteratively. While computationally intensive, LOOCV maximizes data utilization, critical for small datasets where random splits in standard train-test splits might lead to high model variance due to limited sample diversity.

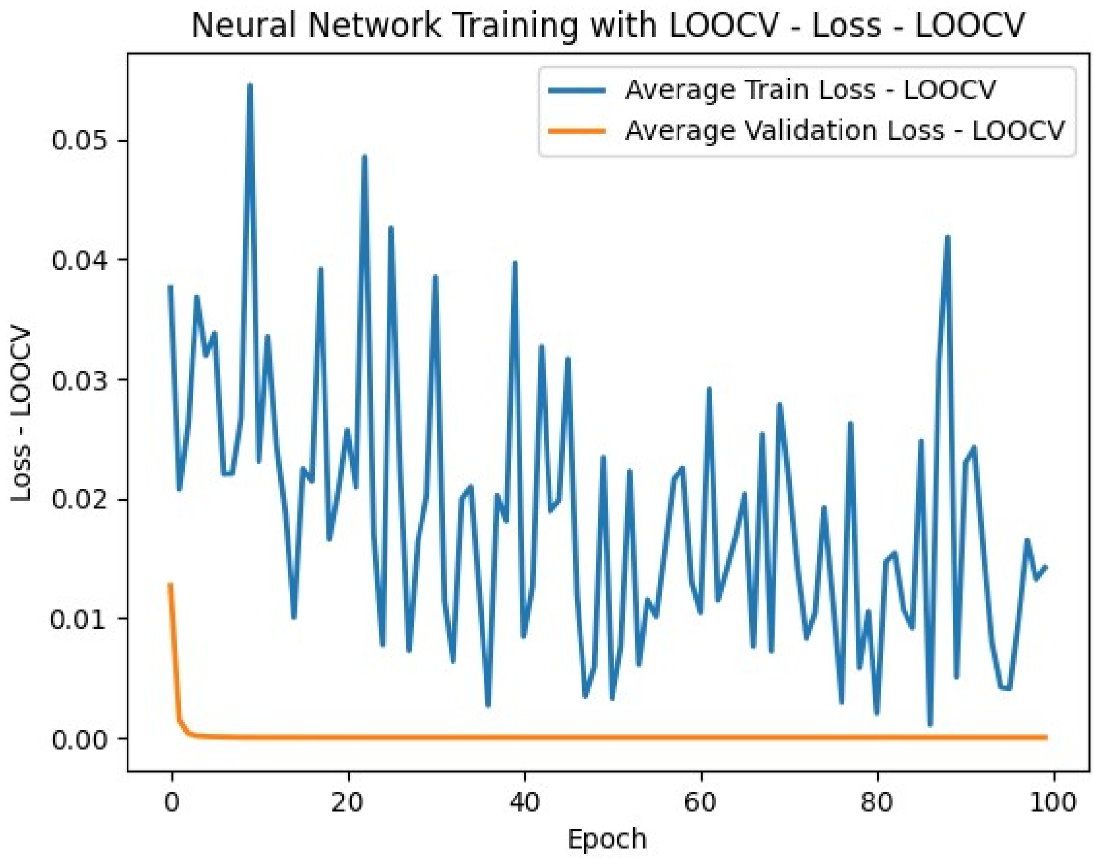
```
from sklearn.model_selection import LeaveOneOut

def nn_loocv(X, y, epochs=100, batch_size=64):
    # Standardize the input features
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # Convert y to a NumPy array for indexing
    y_array = y.to_numpy()

    # Calculate class weights based on class distribution
    class_weights = {0: 1.5405405405405406, 1: 2.85}

    # Build the neural network model
    NN_model = keras.Sequential([
        layers.Dense(64, activation='relu',
```

```
input_shape=X_scaled.shape[1:]),
        layers.Dropout(0.5),
        layers.Dense(32, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(1, activation='sigmoid')
    ])

    # Compile the model with accuracy as the metric
    NN_model.compile(
        loss='binary_crossentropy',
        optimizer=keras.optimizers.Adam(learning_rate=0.001),
        metrics=['accuracy']
    )

    # Initialize LeaveOneOut
    loo = LeaveOneOut()

    # Initialize lists to store loss and accuracy for each fold
    all_train_loss = []
    all_val_loss = []
    all_train_accuracy = []
    all_val_accuracy = []

    # Train and evaluate the model using LOOCV
    for fold_num, (train_index, val_index) in
enumerate(loo.split(X_scaled), 1):
        print(f"Training Fold {fold_num}/{X_scaled.shape[0]} for
{epochs} epochs...")

        # Split the data into training and validation sets
        X_train, X_val = X_scaled[train_index], X_scaled[val_index]
        y_train, y_val = y_array[train_index], y_array[val_index]

        # Train the model
        history = NN_model.fit(
            X_train, y_train,
            validation_data=(X_val, y_val),
class_weight=class_weights,
            batch_size=batch_size,
            epochs=epochs,
            verbose=0  # Set verbose to 0 to suppress output during
training
        )

        # Append loss and accuracy values to lists
        all_train_loss.append(history.history['loss'])
        all_val_loss.append(history.history['val_loss'])
        all_train_accuracy.append(history.history['accuracy'])
        all_val_accuracy.append(history.history['val_accuracy'])
```

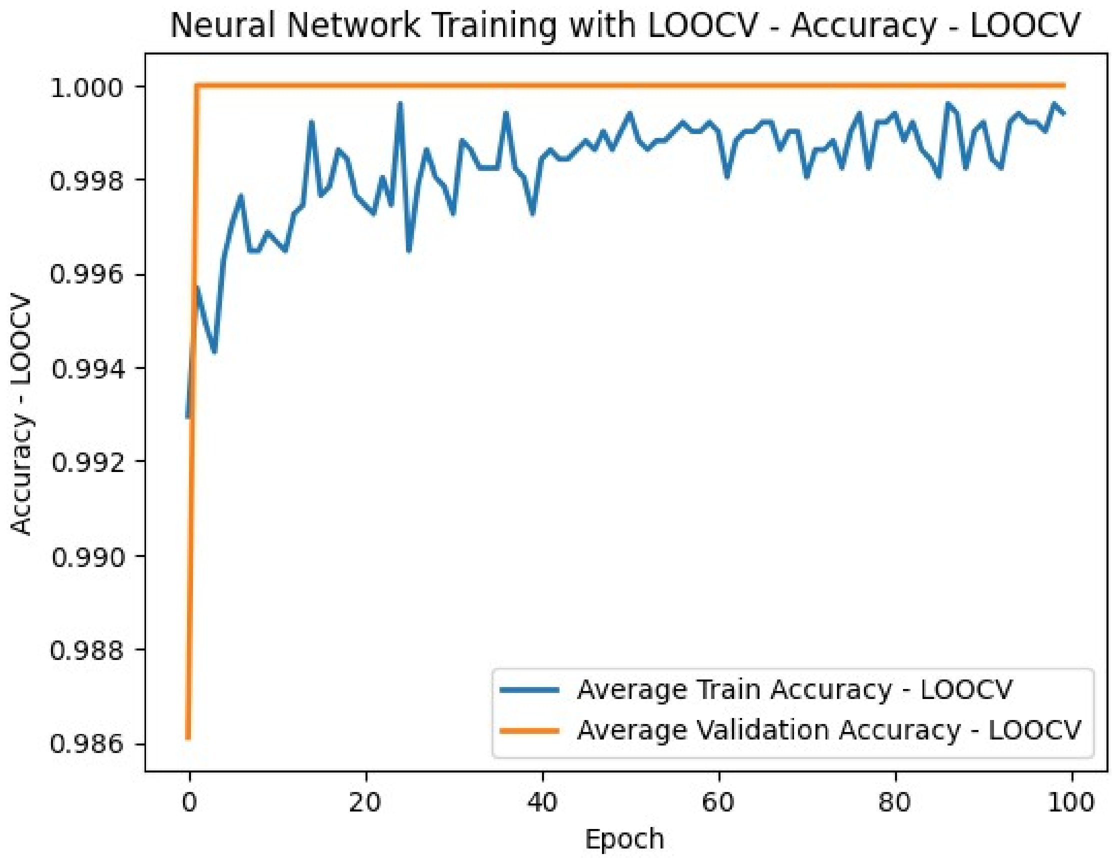
```
    # Plot the average training history over all folds
    plot_avg_hist(all_train_loss, all_val_loss, "Loss - LOOCV")
    plot_avg_hist(all_train_accuracy, all_val_accuracy, "Accuracy -
LOOCV")

def plot_avg_hist(all_train_values, all_val_values, metric_name):
    # Calculate average values over all folds
    avg_train_values = np.mean(all_train_values, axis=0)
    avg_val_values = np.mean(all_val_values, axis=0)

    # Plot the average training history
    plt.plot(avg_train_values, label="Average Train " + metric_name,
lw=2)
    plt.plot(avg_val_values, label="Average Validation " +
metric_name, lw=2)

    # Plot settings
    plt.xlabel("Epoch")
    plt.ylabel(metric_name)
    plt.title(f"Neural Network Training with LOOCV - {metric_name}")
    plt.legend()
    plt.show()

# Example usage
nn_loocv(X, y)
```

```
Training Fold 1/72 for 100 epochs...
Training Fold 2/72 for 100 epochs...
Training Fold 3/72 for 100 epochs...
Training Fold 4/72 for 100 epochs...
Training Fold 5/72 for 100 epochs...
Training Fold 6/72 for 100 epochs...
Training Fold 7/72 for 100 epochs...
Training Fold 8/72 for 100 epochs...
Training Fold 9/72 for 100 epochs...
Training Fold 10/72 for 100 epochs...
Training Fold 11/72 for 100 epochs...
Training Fold 12/72 for 100 epochs...
Training Fold 13/72 for 100 epochs...
Training Fold 14/72 for 100 epochs...
Training Fold 15/72 for 100 epochs...
Training Fold 16/72 for 100 epochs...
Training Fold 17/72 for 100 epochs...
Training Fold 18/72 for 100 epochs...
Training Fold 19/72 for 100 epochs...
Training Fold 20/72 for 100 epochs...
Training Fold 21/72 for 100 epochs...
Training Fold 22/72 for 100 epochs...
Training Fold 23/72 for 100 epochs...
Training Fold 24/72 for 100 epochs...
```

```
Training Fold 25/72 for 100 epochs...
Training Fold 26/72 for 100 epochs...
Training Fold 27/72 for 100 epochs...
Training Fold 28/72 for 100 epochs...
Training Fold 29/72 for 100 epochs...
Training Fold 30/72 for 100 epochs...
Training Fold 31/72 for 100 epochs...
Training Fold 32/72 for 100 epochs...
Training Fold 33/72 for 100 epochs...
Training Fold 34/72 for 100 epochs...
Training Fold 35/72 for 100 epochs...
Training Fold 36/72 for 100 epochs...
Training Fold 37/72 for 100 epochs...
Training Fold 38/72 for 100 epochs...
Training Fold 39/72 for 100 epochs...
Training Fold 40/72 for 100 epochs...
Training Fold 41/72 for 100 epochs...
Training Fold 42/72 for 100 epochs...
Training Fold 43/72 for 100 epochs...
Training Fold 44/72 for 100 epochs...
Training Fold 45/72 for 100 epochs...
Training Fold 46/72 for 100 epochs...
Training Fold 47/72 for 100 epochs...
Training Fold 48/72 for 100 epochs...
Training Fold 49/72 for 100 epochs...
Training Fold 50/72 for 100 epochs...
Training Fold 51/72 for 100 epochs...
Training Fold 52/72 for 100 epochs...
Training Fold 53/72 for 100 epochs...
Training Fold 54/72 for 100 epochs...
Training Fold 55/72 for 100 epochs...
Training Fold 56/72 for 100 epochs...
Training Fold 57/72 for 100 epochs...
Training Fold 58/72 for 100 epochs...
Training Fold 59/72 for 100 epochs...
Training Fold 60/72 for 100 epochs...
Training Fold 61/72 for 100 epochs...
Training Fold 62/72 for 100 epochs...
Training Fold 63/72 for 100 epochs...
Training Fold 64/72 for 100 epochs...
Training Fold 65/72 for 100 epochs...
Training Fold 66/72 for 100 epochs...
Training Fold 67/72 for 100 epochs...
Training Fold 68/72 for 100 epochs...
Training Fold 69/72 for 100 epochs...
Training Fold 70/72 for 100 epochs...
Training Fold 71/72 for 100 epochs...
Training Fold 72/72 for 100 epochs...
```



Neural Network Training with LOOCV - Loss - LOOCV
Average Train Loss - LOOCV
Average Validation Loss - LOOCV
Loss - LOOCV
0.00
0.01
0.02
0.03
0.04
0.05
0
20
40
60
80
100
Epoch



Rationale

K-Fold CV provides a balance between robust evaluation and computational efficiency. By partitioning the dataset into multiple folds, it captures variations in the data distribution and helps detect potential overfitting. The average performance over multiple folds offers a reliable estimate of the model's generalization. However, it may be computationally expensive, especially with large datasets, and the choice of the number of folds (k) can influence results.

On the other hand, LOOCV maximizes data utilization by systematically leaving out a single sample for validation, providing an exhaustive assessment with minimal bias. This approach is particularly beneficial for small datasets, ensuring each data point contributes to model evaluation. However, LOOCV is computationally demanding, especially with larger datasets, and may be sensitive to outliers.