WEEK 2 – EXPERIMENTING WITH DEEP NEURAL NETWORKS

Q1) The data set has 14 features which are as follows:-

1. RowNumber:- Represents the number of rows

2. CustomerId:- Represents customerId

3. Surname:- Represents surname of the customer

4. CreditScore:- Represents credit score of the customer

5. Geography:- Represents the city to which customers belongs to

6. Gender:- Represents Gender of the customer

7. Age:- Represents age of the customer

8. Tenure:- Represents tenure of the customer with a bank

9. Balance:- Represents balance hold by the customer

10. NumOfProducts:- Represents the number of bank services used by the customer

11. HasCrCard:- Represents if a customer has a credit card or not

12. IsActiveMember:- Represents if a customer is an active member or not

13. EstimatedSalary:- Represents estimated salary of the customer

14. Exited:- Represents if a customer is going to exit the bank or not.

1. Perform the required pre-processing and write comment lines to explain the pre-processing steps.

2. Perform experiments using (70,15,15) split and tabulate the performance in terms of Accuracy, Precision

& Recall for the following experimental setup :

1. Number of Hidden Layers and Number of Units per Layer

	Number of Hidden	Layers	Number of Units
1	128	0	0
2	128	64	0
3	128	64	32

2. Epochs (10,20,30)

3. Activation function (Sigmoid)

4. Without Regularization, with Regularization (L1/L2)

5. Learning rate (0.1, 0.01,0.001)

6. Visualize the training and validation loss against the epochs and comment on optimal hyperparameters.

Code:

```
import numpy as np

import pandas as pd

df=pd.read_csv('Churn_Modelling.csv')

df

df.isnull().sum()

#using label encoder to encode categorical columns like gender and geography and dropping unnecessary columns

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

label1 = le.fit_transform(df['Geography'])

label2=le.fit_transform(df['Gender'])

df.drop("Geography", axis=1, inplace=True)

df.drop("RowNumber", axis=1, inplace=True)

df.drop("CustomerId", axis=1, inplace=True)

df.drop("Gender", axis=1, inplace=True)

df.drop("Surname", axis=1, inplace=True)

df["Geography"] = label1

df["Gender"]=label2

df

#defining minmax scaling function and scaling the columns balance,creditscore,EstimatedSalary

def min_max_scaling(column):

    return (column-column.min())/(column.max()-column.min())


df['Balance']=min_max_scaling(df['Balance'])

df['EstimatedSalary']=min_max_scaling(df['EstimatedSalary'])

df['CreditScore']=min_max_scaling(df['CreditScore'])

df
```

```
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense, Input
df['Exit']=df['Exited']
df.drop('Exited',axis=1,inplace=True)
df
X = df.iloc[:, :10]
y = df.iloc[:, 10]
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.15, random_state=40)
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import accuracy_score, precision_score, recall_score
import matplotlib.pyplot as plt
epochs =[10,20,30]
learning_rate =[0.1,0.01,0.001]
units_per_layer = [(128, 64, 32),(128,64,0),(128,0,0)]
# Initialize lists to store performance metrics
results = []

# Initialize lists to store performance metrics
accuracy_scores = []
precision_scores = []
recall_scores = []

for i in units_per_layer:
  for epoch in epochs:
    for lr in learning_rate:
        # Build the neural network model
        model = Sequential([
            Input(shape=X_train.shape[1]),
```

```
        Dense(i[0], activation='relu'),
        Dense(i[1], activation='relu'),
        Dense(i[2], activation='relu'),
        Dense(1, activation='sigmoid')
        ])
    # Compile the model
    model.compile(optimizer=Adam(learning_rate=lr),
    loss='binary_crossentropy',metrics=["accuracy"])
    model.summary()
    # Train the model

    batch_size=32
    history=model.fit(X_train, y_train, epochs=epoch, batch_size=batch_size,
validation_split=0.15)
    y_test_pred = model.predict(X_test)
    y_test_pred = (y_test_pred > 0.5).astype(int)
    accuracy = accuracy_score(y_test, y_test_pred)
    precision = precision_score(y_test, y_test_pred)
    recall = recall_score(y_test, y_test_pred)

    results.append([ i, epoch, lr, accuracy, precision, recall])



    fig=plt.figure()
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Validation'], loc='upper left')
```

```
plt.show()
```

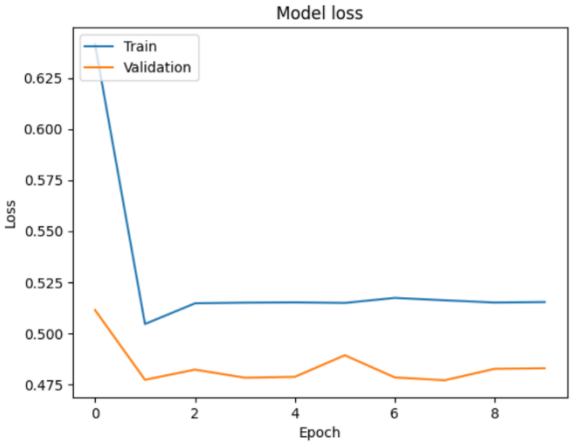
```
columns = ['Units per Layer', 'Epochs', 'Learning Rate', 'Accuracy', 'Precision', 'Recall']
results= pd.DataFrame(results, columns=columns)
results
```

Results & Discussion:

No of units per layer:

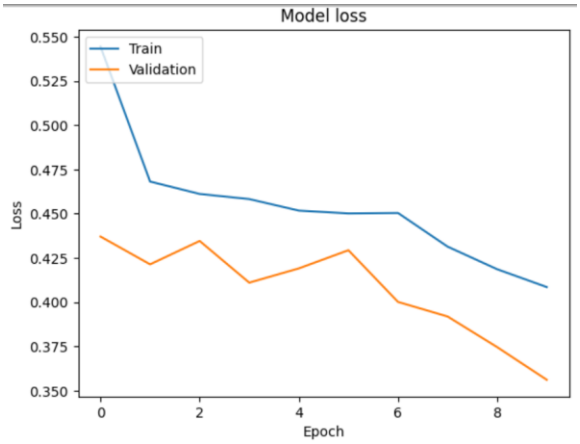
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 128)               1408

 dense_1 (Dense)             (None, 64)                8256

 dense_2 (Dense)             (None, 32)                2080

 dense_3 (Dense)             (None, 1)                 33

=================================================================
Total params: 11777 (46.00 KB)
Trainable params: 11777 (46.00 KB)
Non-trainable params: 0 (0.00 Byte)
```

Learning rate:0.1
Epochs:10



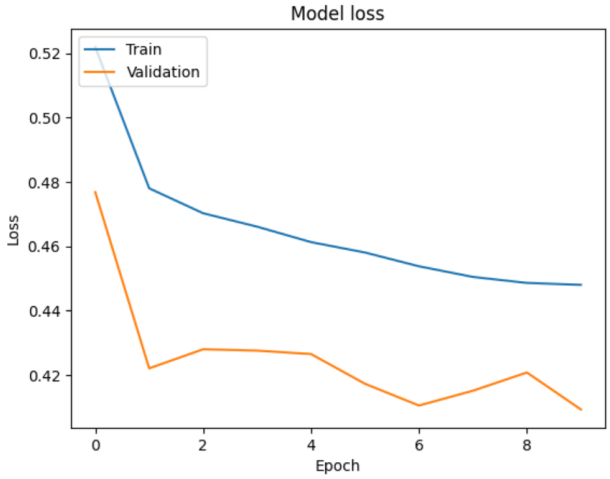
Learning rate:0.01

Epochs:10



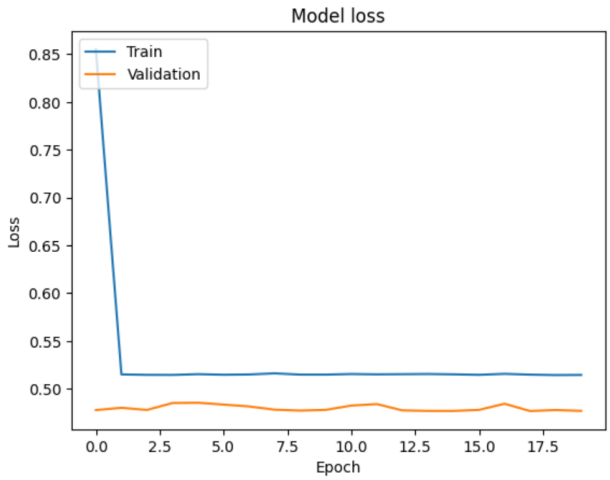
Learning rate:0.001

Epochs:10



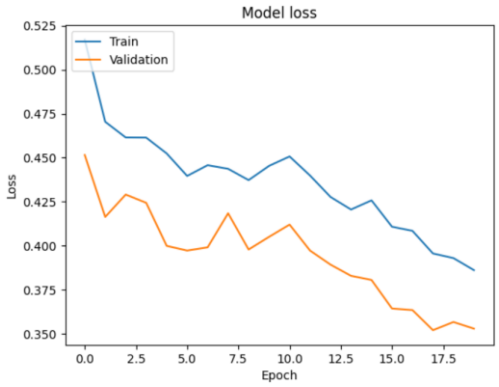
Learning rate:0.1

Epochs:20



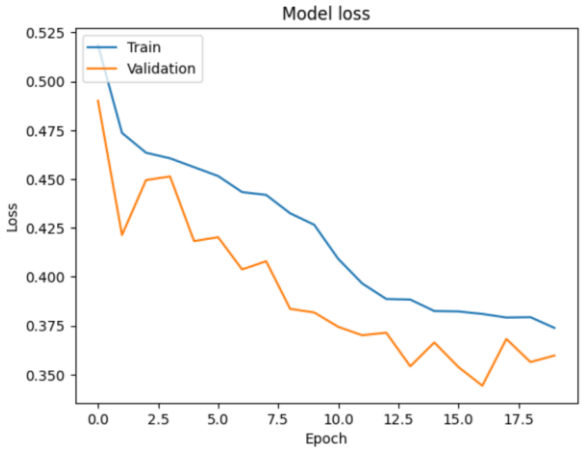
Learning rate:0.01

Epochs:20



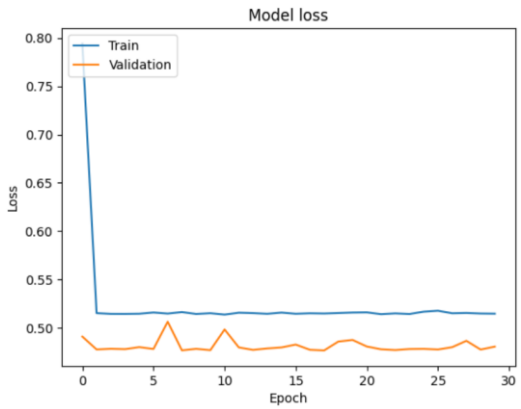
Learning rate:0.001

Epochs:20



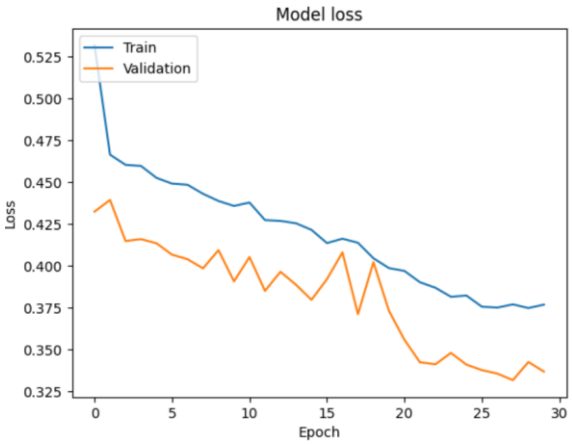
Learning rate:0.1

Epochs:30



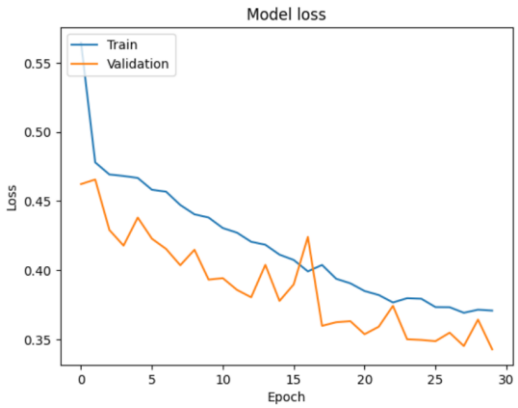
Learning rate:0.01

Epochs:30



Learning rate:0.001

Epochs:30

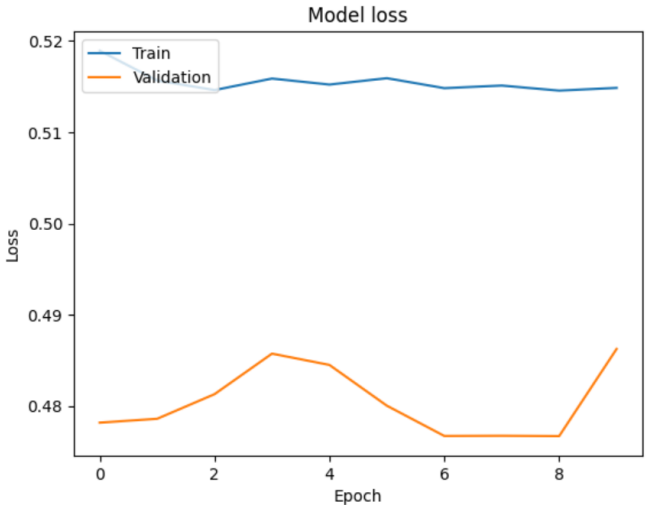


No of units:

```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_36 (Dense)            (None, 128)               1408

 dense_37 (Dense)            (None, 64)                8256

 dense_38 (Dense)            (None, 0)                 0

 dense_39 (Dense)            (None, 1)                 1

=================================================================
Total params: 9665 (37.75 KB)
Trainable params: 9665 (37.75 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

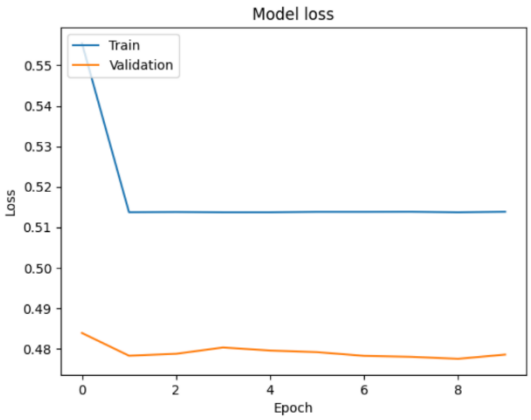
Learning rate:0.1

Epochs:10



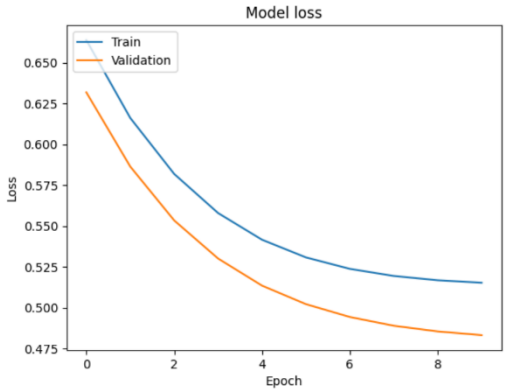
Learning rate:0.01

Epochs:10



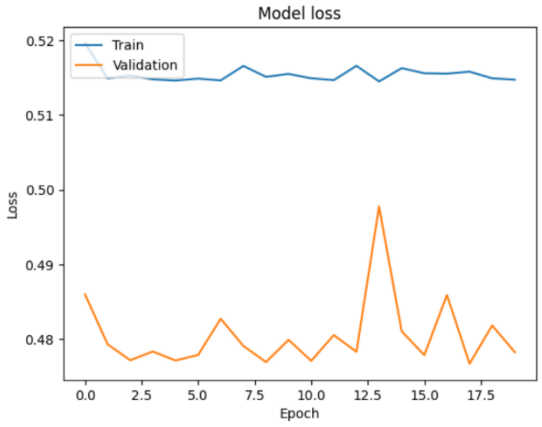
Learning rate:0.001

Epochs:10



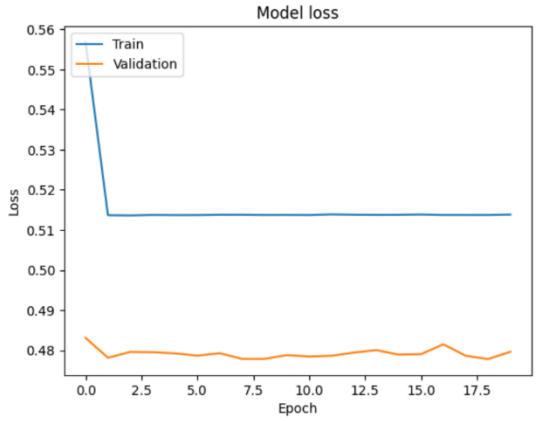
Learning rate:0.1

Epochs:20



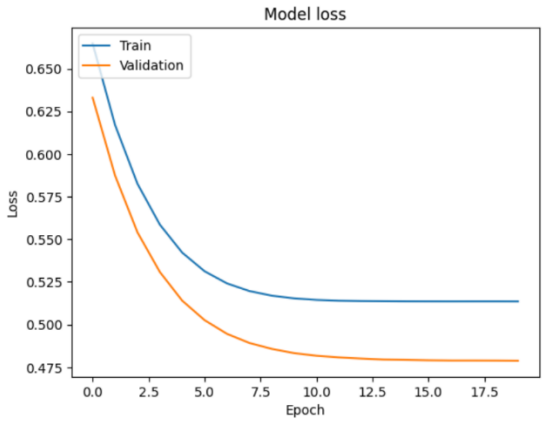
Learning rate:0.01

Epochs:20



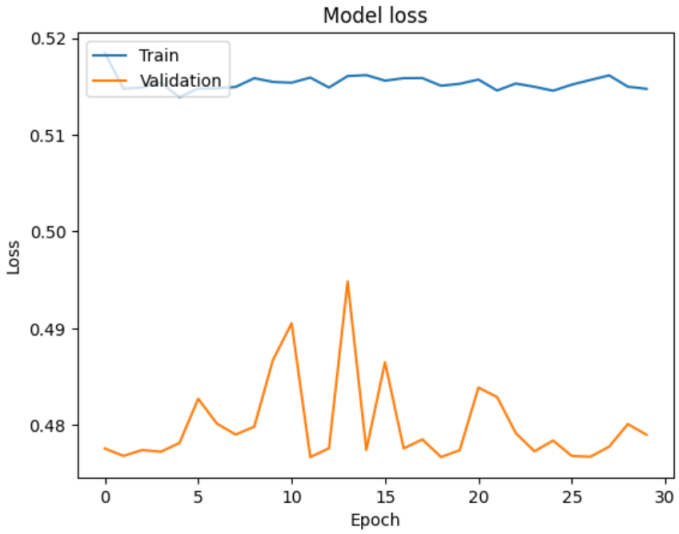
Learning rate:0.001

Epochs:20



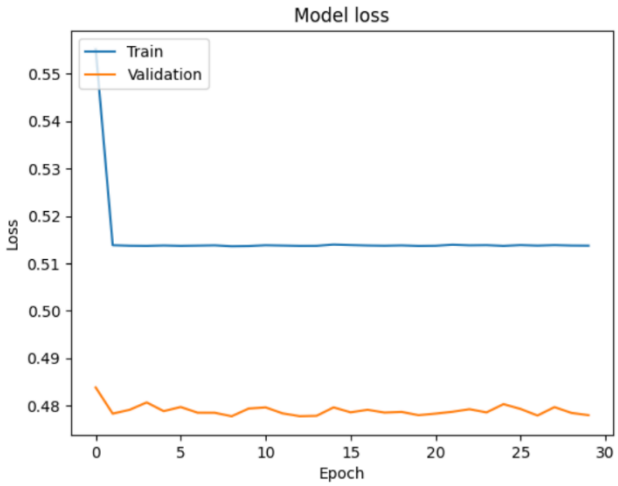
Learning rate:0.1

Epochs:30



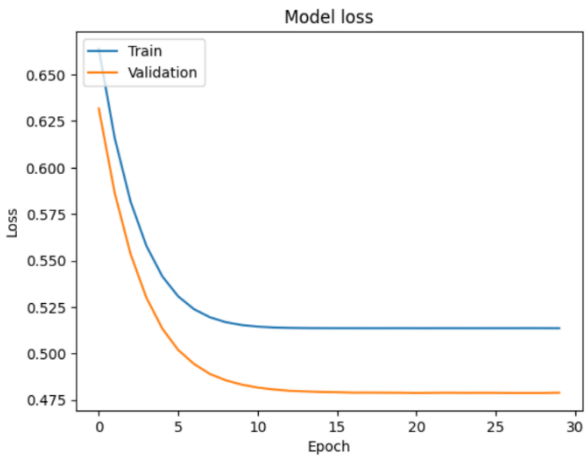
Learning rate:0.01

Epochs:30



Learning rate:0.001

Epochs:30

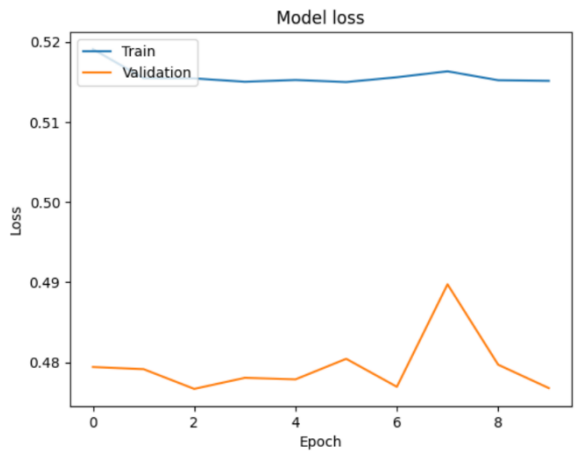


No of units:

```
Model: "sequential_18"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_72 (Dense)            (None, 128)               1408

 dense_73 (Dense)            (None, 0)                 0

 dense_74 (Dense)            (None, 0)                 0

 dense_75 (Dense)            (None, 1)                 1

=================================================================
Total params: 1409 (5.50 KB)
Trainable params: 1409 (5.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

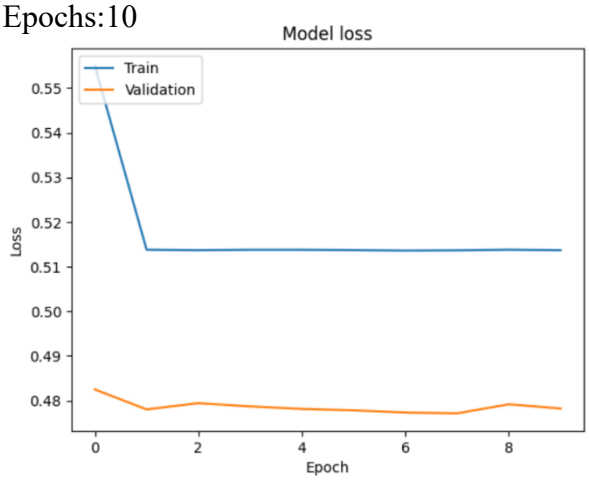
Learning rate:0.1

Epochs:10



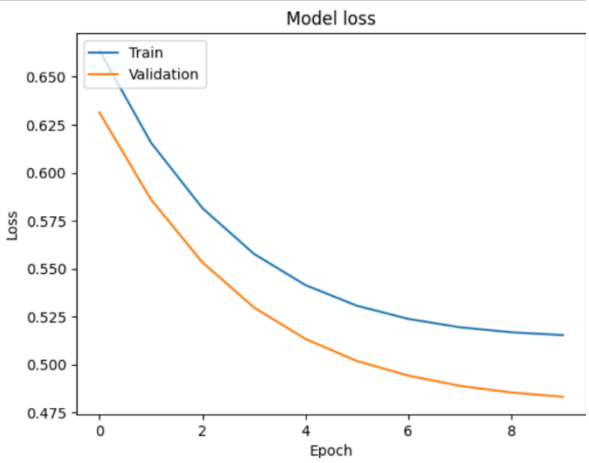
Learning rate:0.01

Epochs:10



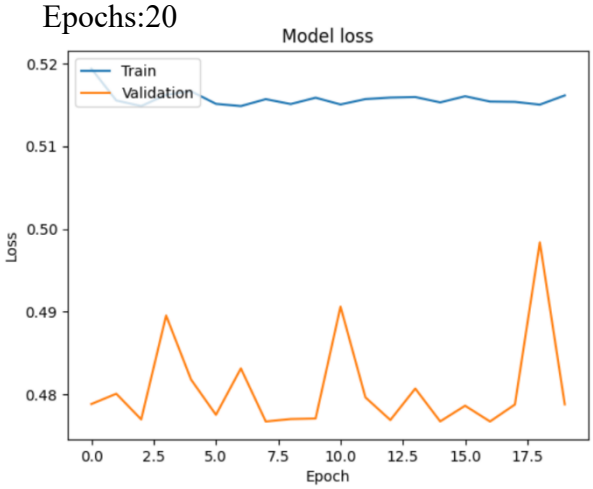
Learning rate:0.001

Epochs:10



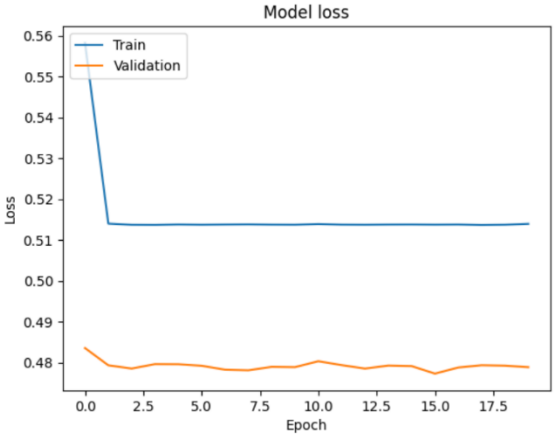
Learning rate:0.1

Epochs:20



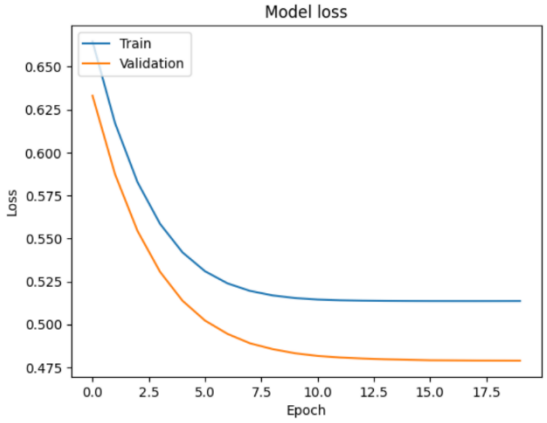
Learning rate:0.01

Epochs:20



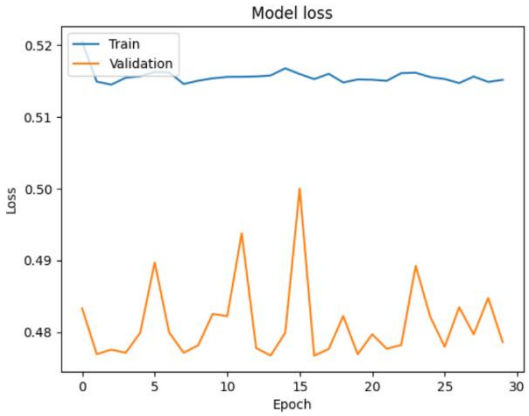
Learning rate:0.001

Epochs:20



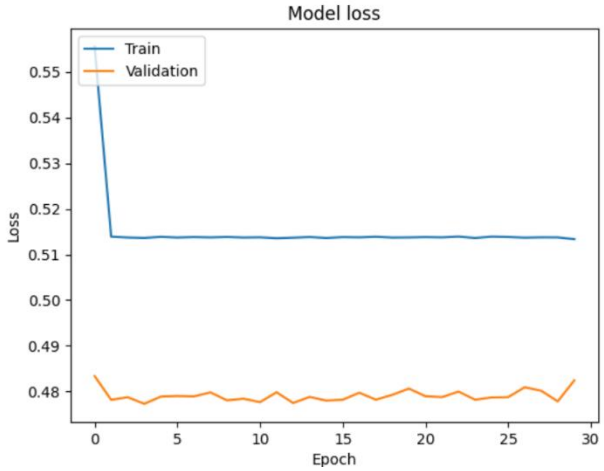
Learning rate:0.1

Epochs:30



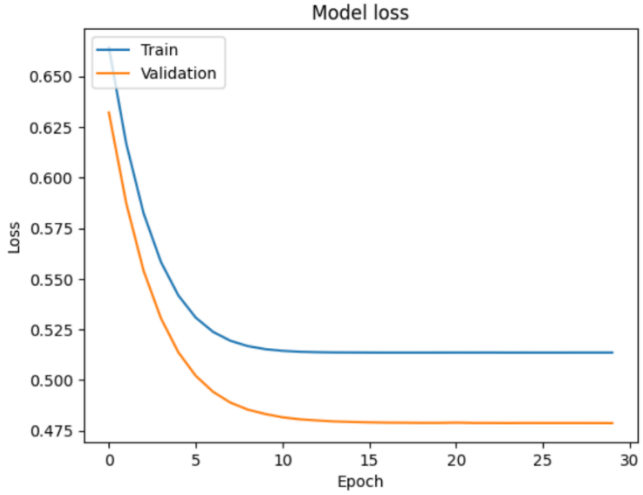
Learning rate:0.01

Epochs:30



Learning rate:0.001

Epochs:30



	Units per Layer	Epochs	Learning Rate	Accuracy	Precision	Recall
0	(128, 64, 32)	10	0.100	0.800000	0.000000	0.000000
1	(128, 64, 32)	10	0.010	0.876000	0.813187	0.493333
2	(128, 64, 32)	10	0.001	0.870000	0.761194	0.510000
3	(128, 64, 32)	20	0.100	0.800000	0.000000	0.000000
4	(128, 64, 32)	20	0.010	0.866667	0.790698	0.453333
5	(128, 64, 32)	20	0.001	0.865333	0.700820	0.570000
6	(128, 64, 32)	30	0.100	0.800000	0.000000	0.000000
7	(128, 64, 32)	30	0.010	0.860000	0.731959	0.473333
8	(128, 64, 32)	30	0.001	0.852667	0.663900	0.533333
9	(128, 64, 0)	10	0.100	0.800000	0.000000	0.000000
10	(128, 64, 0)	10	0.010	0.800000	0.000000	0.000000
11	(128, 64, 0)	10	0.001	0.800000	0.000000	0.000000
12	(128, 64, 0)	20	0.100	0.800000	0.000000	0.000000
13	(128, 64, 0)	20	0.010	0.800000	0.000000	0.000000
14	(128, 64, 0)	20	0.001	0.800000	0.000000	0.000000
15	(128, 64, 0)	30	0.100	0.800000	0.000000	0.000000
16	(128, 64, 0)	30	0.010	0.800000	0.000000	0.000000
17	(128, 64, 0)	30	0.001	0.800000	0.000000	0.000000
18	(128, 0, 0)	10	0.100	0.800000	0.000000	0.000000
19	(128, 0, 0)	10	0.010	0.800000	0.000000	0.000000
20	(128, 0, 0)	10	0.001	0.800000	0.000000	0.000000
21	(128, 0, 0)	20	0.100	0.800000	0.000000	0.000000
22	(128, 0, 0)	20	0.010	0.800000	0.000000	0.000000
23	(128, 0, 0)	20	0.001	0.800000	0.000000	0.000000
24	(128, 0, 0)	30	0.100	0.800000	0.000000	0.000000
25	(128, 0, 0)	30	0.010	0.800000	0.000000	0.000000
26	(128, 0, 0)	30	0.001	0.800000	0.000000	0.000000

Configurations with a learning rate of 0.001 consistently demonstrate better performance across various setups .This suggests that smaller learning rates lead to more stable convergence during training.The architecture with three hidden layers and units (128, 64, 32) per layer generally performs better than architectures with fewer or no hidden units.Deeper networks can capture more intricate patterns in the data, which can be useful for improving model performance.
Many configurations show precision and recall values of 0, which is a concerning sign.
This could be attributed to imbalanced classes, threshold setting, or issues with the model's architecture and training.Configurations with zero units in hidden layers show consistently poor performance with respect to all metrics.This suggests that the model needs more capacity (units) to learn from the data.from the curves, we can tell that some models are overfitting.
In conclusion, the architecture with three hidden layers and units (128, 64, 32) per layer with 0.01 learning rate generally performs better than architectures with fewer or no hidden units.