In [1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.regularizers import l1, l2
from tensorflow.keras.callbacks import EarlyStopping

# Load data
df = pd.read_csv('sonar_csv.csv', header=None)
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
p= df.replace({'Class':{'Rock' : 1, 'Mine' : 0}})
```

In [2]:
```
p
```

Out[2]:

	0	1	2	3	4	5	6	7	
0	attribute_1	attribute_2	attribute_3	attribute_4	attribute_5	attribute_6	attribute_7	attribute_8	at
1	0.02	0.0371	0.0428	0.0207	0.0954	0.0986	0.1539	0.1601	
2	0.0453	0.0523	0.0843	0.0689	0.1183	0.2583	0.2156	0.3481	
3	0.0262	0.0582	0.1099	0.1083	0.0974	0.228	0.2431	0.3771	
4	0.01	0.0171	0.0623	0.0205	0.0205	0.0368	0.1098	0.1276	
...	...	...	...	...	...	...	...	...	
204	0.0187	0.0346	0.0168	0.0177	0.0393	0.163	0.2028	0.1694	
205	0.0323	0.0101	0.0298	0.0564	0.076	0.0958	0.099	0.1018	
206	0.0522	0.0437	0.018	0.0292	0.0351	0.1171	0.1257	0.1178	
207	0.0303	0.0353	0.049	0.0608	0.0167	0.1354	0.1465	0.1123	
208	0.026	0.0363	0.0136	0.0272	0.0214	0.0338	0.0655	0.14	

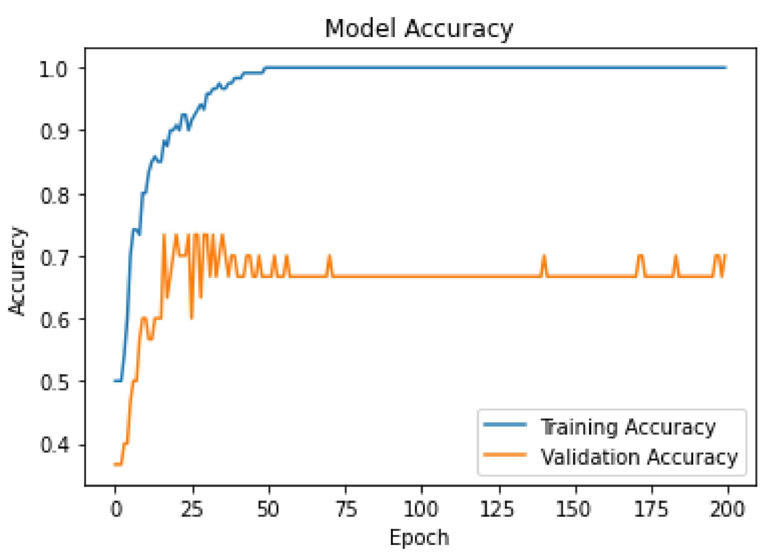
209 rows × 61 columns

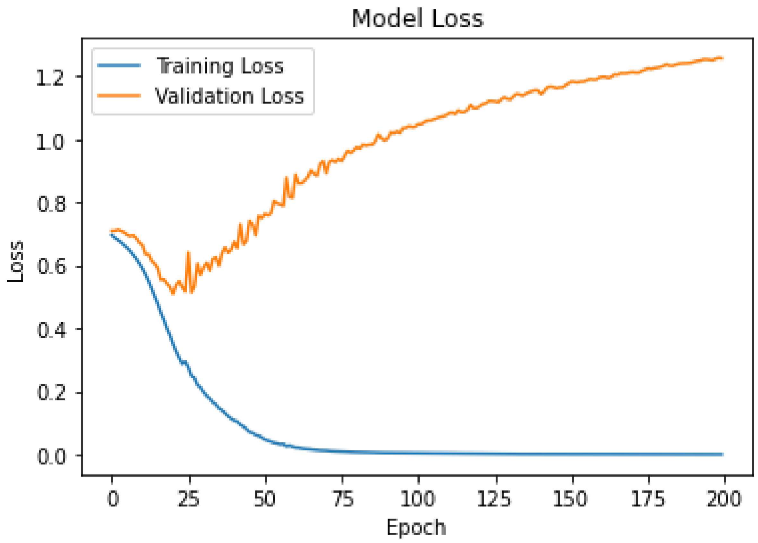
1.)Backpropagation NN with adam optimizer.

In [5]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
import tensorflow as tf

# Load and preprocess the SONAR dataset
sonar_df = pd.read_csv("sonar_csv.csv", header=None)
sonar_df = sonar_df.drop(0)
sonar_df = sonar_df.sample(frac=1).reset_index(drop=True)

X = sonar_df.drop(columns=[60])
y = sonar_df[60]

scaler = MinMaxScaler()
X = scaler.fit_transform(X)

X = np.array(X)
y = tf.keras.utils.to_categorical(y)

train_X, train_y = X[:150], y[:150]
test_X, test_y = X[150:], y[150:]

# Define the neural network architecture
model = tf.keras.Sequential([
  tf.keras.layers.Dense(units=64, activation='relu', input_shape=(60,)),
  tf.keras.layers.Dense(units=32, activation='relu'),
  tf.keras.layers.Dense(units=32, activation='relu'),
  tf.keras.layers.Dense(2, activation='softmax')
])

# Compile the model
model.compile(loss='binary_crossentropy',
              optimizer=tf.keras.optimizers.Adam(),
              metrics=['accuracy'])

# Train the model
history = model.fit(train_X, train_y, epochs=200, validation_split=0.2, verbos
```

```
4/4 [==============================] - 0s 11ms/step - loss: 0.5897 - accur
acy: 0.8000 - val_loss: 0.6641 - val_accuracy: 0.6000
Epoch 12/200
4/4 [==============================] - 0s 10ms/step - loss: 0.5707 - accur
acy: 0.8333 - val_loss: 0.6349 - val_accuracy: 0.5667
Epoch 13/200
4/4 [==============================] - 0s 10ms/step - loss: 0.5495 - accur
acy: 0.8500 - val_loss: 0.6349 - val_accuracy: 0.5667
Epoch 14/200
4/4 [==============================] - 0s 12ms/step - loss: 0.5264 - accur
acy: 0.8583 - val_loss: 0.6142 - val_accuracy: 0.6000
Epoch 15/200
4/4 [==============================] - 0s 10ms/step - loss: 0.5011 - accur
acy: 0.8500 - val_loss: 0.6049 - val_accuracy: 0.6000
Epoch 16/200
4/4 [==============================] - 0s 11ms/step - loss: 0.4781 - accur
acy: 0.8500 - val_loss: 0.5903 - val_accuracy: 0.6000
Epoch 17/200
4/4 [==============================] - 0s 10ms/step - loss: 0.4493 - accur
acy: 0.8833 - val_loss: 0.5534 - val_accuracy: 0.7333
```

In [6]:
```
import matplotlib.pyplot as plt

# Plot training and validation accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plot training and validation loss
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Evaluate the model
loss, accuracy = model.evaluate(test_X, test_y, verbose=0)
print(f"Test Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy:.4f}")
```






```
Test Loss: 1.2187
Test Accuracy: 0.7586
```

In [ ]:

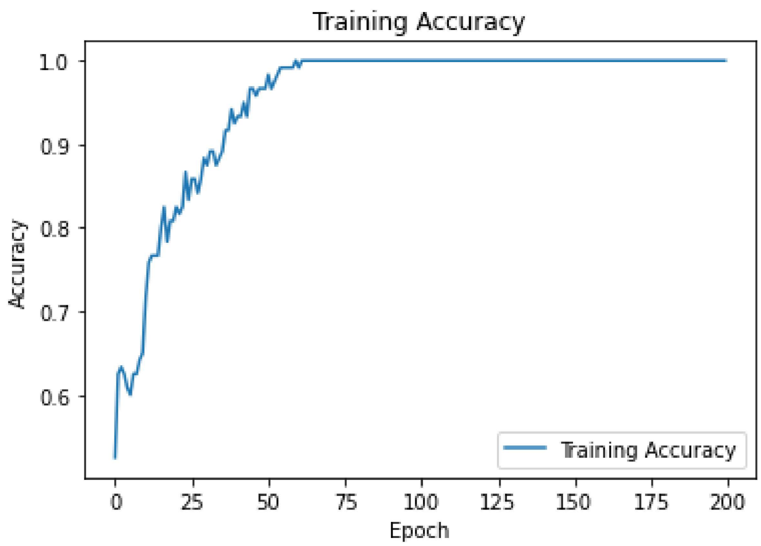
2.)l1 and l2 regularizations.

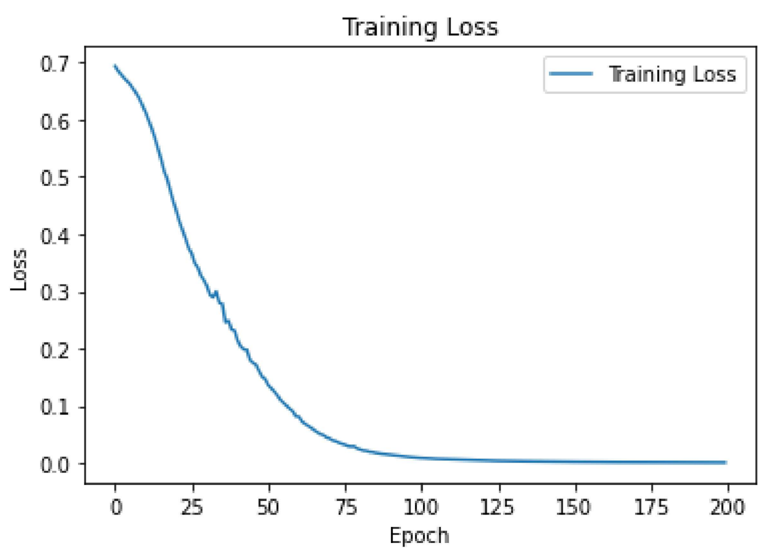
In [11]:

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler

sonar_df = pd.read_csv("sonar_csv.csv", header=None)

sonar_df = sonar_df.drop(0)



sonar_df = sonar_df.sample(frac=1).reset_index(drop=True)
sonar_df
X = sonar_df.drop(columns=[60])
y = sonar_df[60]

scaler = MinMaxScaler()
X = scaler.fit_transform(X)

X = np.array(X)
y = tf.keras.utils.to_categorical(y)  # Convert labels to one-hot encoded form

train_X, train_y = X[:150], y[:150]
test_X, test_y = X[150:], y[150:]

model = tf.keras.Sequential([
  tf.keras.layers.Dense(units=64, activation='relu', input_shape=(60,), kernel
  tf.keras.layers.Dense(units=32, activation='relu'),
  tf.keras.layers.Dense(units=32, activation='relu', kernel_regularizer=tf.ker
  tf.keras.layers.Dense(2, activation='softmax')
])


model.compile(loss='binary_crossentropy',
              optimizer=tf.keras.optimizers.Adam(),
              metrics=['accuracy'])
history = model.fit(train_X, train_y, epochs=200)
```

```
5/5 [==============================] - 0s 4ms/step - loss: 0.6893 - accura
cy: 0.5667
Epoch 129/200
5/5 [==============================] - 0s 5ms/step - loss: 0.6894 - accura
cy: 0.5667
Epoch 130/200
5/5 [==============================] - 0s 4ms/step - loss: 0.6895 - accura
cy: 0.5667
Epoch 131/200
5/5 [==============================] - 0s 4ms/step - loss: 0.6895 - accura
cy: 0.5667
Epoch 132/200
5/5 [==============================] - 0s 4ms/step - loss: 0.6894 - accura
cy: 0.5667
Epoch 133/200
5/5 [==============================] - 0s 4ms/step - loss: 0.6894 - accura
cy: 0.5667
Epoch 134/200
5/5 [==============================] - 0s 6ms/step - loss: 0.6896 - accura
cy: 0.5667
```

In [4]:
```
import matplotlib.pyplot as plt




# Plot the training accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.title('Training Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plot the training loss
plt.plot(history.history['loss'], label='Training Loss')
plt.title('Training Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```





3.)Early stopping for input and second hidden layer.

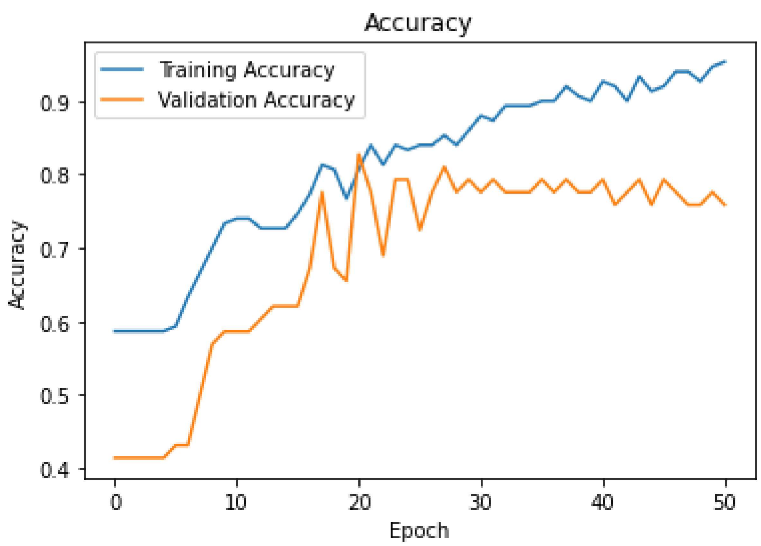
In [7]:
```
import tensorflow as tf
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.callbacks import EarlyStopping

sonar_df = pd.read_csv("sonar_csv.csv", header=None)

sonar_df = sonar_df.drop(0)


sonar_df = sonar_df.sample(frac=1).reset_index(drop=True)

X = sonar_df.drop(columns=[60])
y = sonar_df[60]

scaler = MinMaxScaler()
X = scaler.fit_transform(X)

X = np.array(X)
y = tf.keras.utils.to_categorical(y)

train_X, train_y = X[:150], y[:150]
test_X, test_y = X[150:], y[150:]

early_stopping = EarlyStopping(monitor='val_loss', patience=10)

model = tf.keras.Sequential([
  tf.keras.layers.Dense(units=64, activation='relu', input_shape=(60,)),
  tf.keras.layers.Dense(units=32, activation='relu'),
  tf.keras.layers.Dense(2, activation='softmax')
])

model.compile(loss='binary_crossentropy',
              optimizer=tf.keras.optimizers.Adam(),
              metrics=['accuracy'])

history = model.fit(train_X, train_y,
                    epochs=200,
                    validation_data=(test_X, test_y),
                    callbacks=[early_stopping])
```

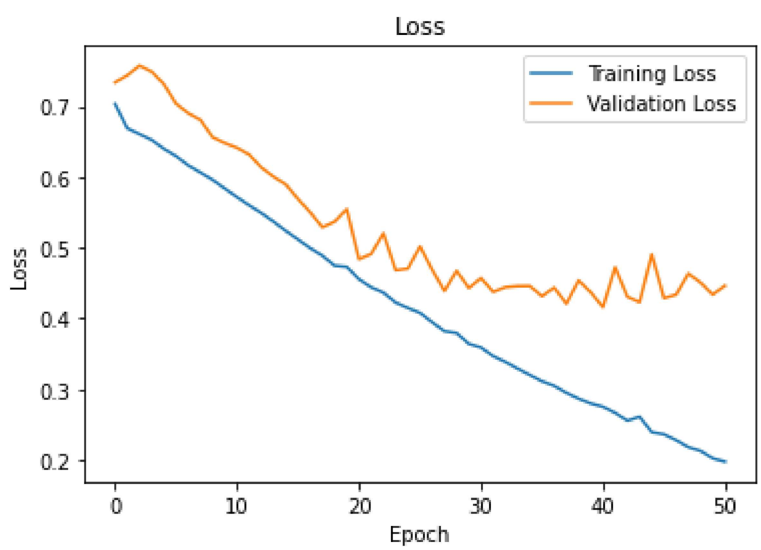
```
Epoch 1/200
5/5 [==============================] - 0s 32ms/step - loss: 0.7034 - accurac
y: 0.5867 - val_loss: 0.7337 - val_accuracy: 0.4138
Epoch 2/200
5/5 [==============================] - 0s 10ms/step - loss: 0.6689 - accurac
y: 0.5867 - val_loss: 0.7437 - val_accuracy: 0.4138
Epoch 3/200
5/5 [==============================] - 0s 8ms/step - loss: 0.6606 - accuracy:
0.5867 - val_loss: 0.7575 - val_accuracy: 0.4138
Epoch 4/200
5/5 [==============================] - 0s 11ms/step - loss: 0.6524 - accurac
y: 0.5867 - val_loss: 0.7489 - val_accuracy: 0.4138
Epoch 5/200
5/5 [==============================] - 0s 8ms/step - loss: 0.6399 - accuracy:
0.5867 - val_loss: 0.7313 - val_accuracy: 0.4138
Epoch 6/200
5/5 [==============================] - 0s 9ms/step - loss: 0.6294 - accuracy:
0.5933 - val_loss: 0.7037 - val_accuracy: 0.4310
Epoch 7/200
5/5 [==============================] - 0s 9ms/step - loss: 0.6164 - accuracy:
0.6333 - val_loss: 0.6902 - val_accuracy: 0.4310
Epoch 8/200
5/5 [==============================] - 0s 8ms/step - loss: 0.6062 - accuracy:
0.6667 - val_loss: 0.6807 - val_accuracy: 0.5000
Epoch 9/200
5/5 [==============================] - 0s 8ms/step - loss: 0.5959 - accuracy:
0.7000 - val_loss: 0.6562 - val_accuracy: 0.5690
Epoch 10/200
5/5 [==============================] - 0s 8ms/step - loss: 0.5836 - accuracy:
0.7333 - val_loss: 0.6480 - val_accuracy: 0.5862
Epoch 11/200
5/5 [==============================] - 0s 9ms/step - loss: 0.5715 - accuracy:
0.7400 - val_loss: 0.6412 - val_accuracy: 0.5862
Epoch 12/200
5/5 [==============================] - 0s 9ms/step - loss: 0.5599 - accuracy:
0.7400 - val_loss: 0.6316 - val_accuracy: 0.5862
Epoch 13/200
5/5 [==============================] - 0s 9ms/step - loss: 0.5490 - accuracy:
0.7267 - val_loss: 0.6133 - val_accuracy: 0.6034
Epoch 14/200
5/5 [==============================] - 0s 8ms/step - loss: 0.5367 - accuracy:
0.7267 - val_loss: 0.6002 - val_accuracy: 0.6207
Epoch 15/200
5/5 [==============================] - 0s 8ms/step - loss: 0.5237 - accuracy:
0.7267 - val_loss: 0.5894 - val_accuracy: 0.6207
Epoch 16/200
5/5 [==============================] - 0s 8ms/step - loss: 0.5115 - accuracy:
0.7467 - val_loss: 0.5689 - val_accuracy: 0.6207
Epoch 17/200
5/5 [==============================] - 0s 9ms/step - loss: 0.4993 - accuracy:
0.7733 - val_loss: 0.5500 - val_accuracy: 0.6724
Epoch 18/200
5/5 [==============================] - 0s 8ms/step - loss: 0.4886 - accuracy:
0.8133 - val_loss: 0.5288 - val_accuracy: 0.7759
Epoch 19/200
5/5 [==============================] - 0s 9ms/step - loss: 0.4745 - accuracy:
0.8067 - val_loss: 0.5368 - val_accuracy: 0.6724
```

```
Epoch 20/200
5/5 [==============================] - 0s 8ms/step - loss: 0.4728 - accuracy:
0.7667 - val_loss: 0.5545 - val_accuracy: 0.6552
Epoch 21/200
5/5 [==============================] - 0s 9ms/step - loss: 0.4553 - accuracy:
0.8067 - val_loss: 0.4837 - val_accuracy: 0.8276
Epoch 22/200
5/5 [==============================] - 0s 8ms/step - loss: 0.4440 - accuracy:
0.8400 - val_loss: 0.4911 - val_accuracy: 0.7759
Epoch 23/200
5/5 [==============================] - 0s 9ms/step - loss: 0.4365 - accuracy:
0.8133 - val_loss: 0.5201 - val_accuracy: 0.6897
Epoch 24/200
5/5 [==============================] - 0s 9ms/step - loss: 0.4224 - accuracy:
0.8400 - val_loss: 0.4680 - val_accuracy: 0.7931
Epoch 25/200
5/5 [==============================] - 0s 8ms/step - loss: 0.4147 - accuracy:
0.8333 - val_loss: 0.4702 - val_accuracy: 0.7931
Epoch 26/200
5/5 [==============================] - 0s 8ms/step - loss: 0.4078 - accuracy:
0.8400 - val_loss: 0.5017 - val_accuracy: 0.7241
Epoch 27/200
5/5 [==============================] - 0s 10ms/step - loss: 0.3944 - accurac
y: 0.8400 - val_loss: 0.4685 - val_accuracy: 0.7759
Epoch 28/200
5/5 [==============================] - 0s 9ms/step - loss: 0.3818 - accuracy:
0.8533 - val_loss: 0.4388 - val_accuracy: 0.8103
Epoch 29/200
5/5 [==============================] - 0s 9ms/step - loss: 0.3792 - accuracy:
0.8400 - val_loss: 0.4671 - val_accuracy: 0.7759
Epoch 30/200
5/5 [==============================] - 0s 8ms/step - loss: 0.3639 - accuracy:
0.8600 - val_loss: 0.4430 - val_accuracy: 0.7931
Epoch 31/200
5/5 [==============================] - 0s 9ms/step - loss: 0.3587 - accuracy:
0.8800 - val_loss: 0.4568 - val_accuracy: 0.7759
Epoch 32/200
5/5 [==============================] - 0s 9ms/step - loss: 0.3465 - accuracy:
0.8733 - val_loss: 0.4375 - val_accuracy: 0.7931
Epoch 33/200
5/5 [==============================] - 0s 10ms/step - loss: 0.3383 - accurac
y: 0.8933 - val_loss: 0.4441 - val_accuracy: 0.7759
Epoch 34/200
5/5 [==============================] - 0s 7ms/step - loss: 0.3289 - accuracy:
0.8933 - val_loss: 0.4457 - val_accuracy: 0.7759
Epoch 35/200
5/5 [==============================] - 0s 8ms/step - loss: 0.3200 - accuracy:
0.8933 - val_loss: 0.4458 - val_accuracy: 0.7759
Epoch 36/200
5/5 [==============================] - 0s 8ms/step - loss: 0.3111 - accuracy:
0.9000 - val_loss: 0.4314 - val_accuracy: 0.7931
Epoch 37/200
5/5 [==============================] - 0s 13ms/step - loss: 0.3046 - accurac
y: 0.9000 - val_loss: 0.4435 - val_accuracy: 0.7759
Epoch 38/200
5/5 [==============================] - 0s 9ms/step - loss: 0.2946 - accuracy:
0.9200 - val_loss: 0.4208 - val_accuracy: 0.7931
```

```
Epoch 39/200
5/5 [==============================] - 0s 9ms/step - loss: 0.2864 - accuracy:
0.9067 - val_loss: 0.4536 - val_accuracy: 0.7759
Epoch 40/200
5/5 [==============================] - 0s 8ms/step - loss: 0.2796 - accuracy:
0.9000 - val_loss: 0.4369 - val_accuracy: 0.7759
Epoch 41/200
5/5 [==============================] - 0s 8ms/step - loss: 0.2749 - accuracy:
0.9267 - val_loss: 0.4157 - val_accuracy: 0.7931
Epoch 42/200
5/5 [==============================] - 0s 9ms/step - loss: 0.2663 - accuracy:
0.9200 - val_loss: 0.4722 - val_accuracy: 0.7586
Epoch 43/200
5/5 [==============================] - 0s 7ms/step - loss: 0.2552 - accuracy:
0.9000 - val_loss: 0.4306 - val_accuracy: 0.7759
Epoch 44/200
5/5 [==============================] - 0s 8ms/step - loss: 0.2608 - accuracy:
0.9333 - val_loss: 0.4229 - val_accuracy: 0.7931
Epoch 45/200
5/5 [==============================] - 0s 7ms/step - loss: 0.2389 - accuracy:
0.9133 - val_loss: 0.4905 - val_accuracy: 0.7586
Epoch 46/200
5/5 [==============================] - 0s 8ms/step - loss: 0.2359 - accuracy:
0.9200 - val_loss: 0.4286 - val_accuracy: 0.7931
Epoch 47/200
5/5 [==============================] - 0s 9ms/step - loss: 0.2276 - accuracy:
0.9400 - val_loss: 0.4336 - val_accuracy: 0.7759
Epoch 48/200
5/5 [==============================] - 0s 8ms/step - loss: 0.2177 - accuracy:
0.9400 - val_loss: 0.4633 - val_accuracy: 0.7586
Epoch 49/200
5/5 [==============================] - 0s 9ms/step - loss: 0.2125 - accuracy:
0.9267 - val_loss: 0.4506 - val_accuracy: 0.7586
Epoch 50/200
5/5 [==============================] - 0s 9ms/step - loss: 0.2019 - accuracy:
0.9467 - val_loss: 0.4336 - val_accuracy: 0.7759
Epoch 51/200
5/5 [==============================] - 0s 8ms/step - loss: 0.1973 - accuracy:
0.9533 - val_loss: 0.4462 - val_accuracy: 0.7586
```

In [8]:
```
import matplotlib.pyplot as plt

# Plot the training and validation accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plot the training and validation loss
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```





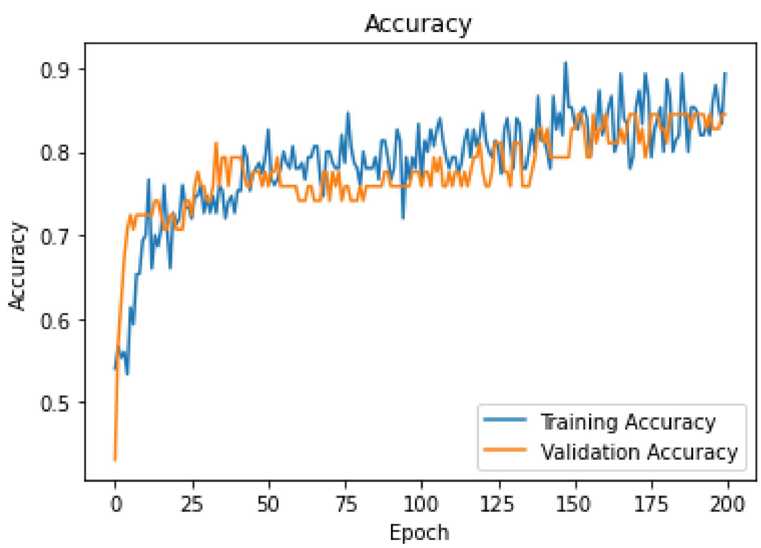
In [ ]:

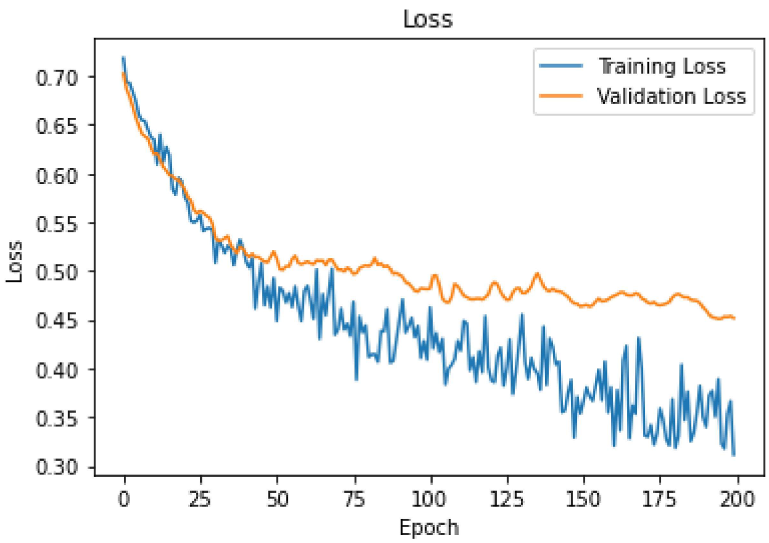
4.)Dropout regularization with p = 0.2 for input and second hidden layer.

In [9]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
import tensorflow as tf

sonar_df = pd.read_csv("sonar_csv.csv", header=None)
sonar_df = sonar_df.drop(0)
sonar_df = sonar_df.sample(frac=1).reset_index(drop=True)

X = sonar_df.drop(columns=[60])
y = sonar_df[60]

scaler = MinMaxScaler()
X = scaler.fit_transform(X)

X = np.array(X)
y = tf.keras.utils.to_categorical(y)

train_X, train_y = X[:150], y[:150]
test_X, test_y = X[150:], y[150:]

model = tf.keras.Sequential([
  tf.keras.layers.Dropout(0.2, input_shape=(60,)),  # Dropout layer as the inp
  tf.keras.layers.Dense(units=64, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(2, activation='softmax')
])

model.compile(loss='binary_crossentropy',
              optimizer=tf.keras.optimizers.Adam(),
              metrics=['accuracy'])

history = model.fit(train_X, train_y, epochs=200, validation_data=(test_X, tes
```

```
5/5 [==============================] - 0s 13ms/step - loss: 0.3504 - accur
acy: 0.8333 - val_loss: 0.4519 - val_accuracy: 0.8276
Epoch 195/200
5/5 [==============================] - 0s 14ms/step - loss: 0.3889 - accur
acy: 0.8200 - val_loss: 0.4509 - val_accuracy: 0.8448
Epoch 196/200
5/5 [==============================] - 0s 13ms/step - loss: 0.3231 - accur
acy: 0.8600 - val_loss: 0.4512 - val_accuracy: 0.8276
Epoch 197/200
5/5 [==============================] - 0s 13ms/step - loss: 0.3177 - accur
acy: 0.8800 - val_loss: 0.4530 - val_accuracy: 0.8276
Epoch 198/200
5/5 [==============================] - 0s 12ms/step - loss: 0.3538 - accur
acy: 0.8533 - val_loss: 0.4527 - val_accuracy: 0.8276
Epoch 199/200
5/5 [==============================] - 0s 10ms/step - loss: 0.3665 - accur
acy: 0.8333 - val_loss: 0.4536 - val_accuracy: 0.8448
Epoch 200/200
5/5 [==============================] - 0s 9ms/step - loss: 0.3117 - accura
cy: 0.8933 - val_loss: 0.4517 - val_accuracy: 0.8448
```

In [10]:
```
import matplotlib.pyplot as plt


# Plot the training and validation accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plot the training and validation loss
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```




Loss
Training Loss
Validation Loss
Loss
Epoch
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0.30
0
25
50
75
100
125
150
175
200

In [ ]: