Problem 2

Using the Wisconsin Breast Cancer (Diagnostic) dataset from Scikit-learn featured in previous homework assignments, three machine learning models (a Connected Neural Network (NN), Logistic Regression (LR), and a Support Vector Machine (SVM)) will be implemented and the results analyzed.

```
import os
import random
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report, confusion_matrix, ro
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
import tensorflow as tf
```

```
# Reproducibility
seed = 42
np.random.seed(seed)
random.seed(seed)
tf.random.set_seed(seed)
```

The dataset, consisting of 569 instances and 30 numerical features derived from tumor cell nuclei characteristics, was used.

```
# Load the sklearn breast cancer dataset
from sklearn.datasets import load_breast_cancer

sk = load_breast_cancer()
X = sk.data
y = sk.target
feature_names = sk.feature_names

print("X shape:", X.shape, "y shape:", y.shape)
```

```
X shape: (569, 30) y shape: (569,)
```

All 30 features were standardized using `StandardScaler`, fitting only on the training data to prevent data leakage.

```
# Train/validation split (80% train / 20% val)
X_train, X_val, y_train, y_val = train_test_split(
    X, y, test_size=0.20, random_state=seed, stratify=y
)

# Standardize features (fit on train only)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)
```

The Fully Connected Neural Network was designed with three dense hidden layers, incorporating Batch Normalization and Dropout to improve stability and prevent overfitting.

```
# Build fully connected neural network (multiple hidden layers)
keras = tf.keras
layers = tf.keras.layers
Sequential = tf.keras.Sequential
Input = tf.keras.Input
Dense = tf.keras.layers.Dense
BatchNormalization = tf.keras.layers.BatchNormalization
Dropout = tf.keras.layers.Dropout

def build_model(input_dim):
    model = Sequential([
        Input(shape=(input_dim,)),
        Dense(128, activation='relu'),
```

```
        BatchNormalization(),
        Dropout(0.3),
        Dense(64, activation='relu'),
        BatchNormalization(),
        Dropout(0.2),
        Dense(32, activation='relu'),
        Dense(1, activation='sigmoid')
    ])
    return model

model = build_model(X_train.shape[1])
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
    loss='binary_crossentropy',
    metrics=['accuracy']
)
model.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
dense_8 (Dense)	(None, 128)	3,968
batch_normalization_4 (BatchNormalization)	(None, 128)	512
dropout_4 (Dropout)	(None, 128)	0
dense_9 (Dense)	(None, 64)	8,256
batch_normalization_5 (BatchNormalization)	(None, 64)	256
dropout_5 (Dropout)	(None, 64)	0
dense_10 (Dense)	(None, 32)	2,080
dense_11 (Dense)	(None, 1)	33

Total params: 15,105 (59.00 KB)
Trainable params: 14,721 (57.50 KB)
Non-trainable params: 384 (1.50 KB)

An Early Stopping callback with a patience of 15 epochs was used to halt training and restore the best weights, ensuring optimal generalization.

```
# Train the model with EarlyStopping
callbacks = [
    tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=15, restore_best_weights=True)
]

history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=200,
    batch_size=32,
    callbacks=callbacks,
    verbose=2
)
```

```
Epoch 5/200
15/15 - 0s - 8ms/step - accuracy: 0.9736 - loss: 0.1025 - val_accuracy: 0.9561 - val_loss: 0.1862
Epoch 6/200
15/15 - 0s - 8ms/step - accuracy: 0.9714 - loss: 0.0926 - val_accuracy: 0.9561 - val_loss: 0.1581
Epoch 7/200
15/15 - 0s - 8ms/step - accuracy: 0.9846 - loss: 0.0762 - val_accuracy: 0.9561 - val_loss: 0.1394
Epoch 8/200
15/15 - 0s - 8ms/step - accuracy: 0.9758 - loss: 0.0719 - val_accuracy: 0.9561 - val_loss: 0.1259
Epoch 9/200
15/15 - 0s - 9ms/step - accuracy: 0.9802 - loss: 0.0752 - val_accuracy: 0.9561 - val_loss: 0.1144
```

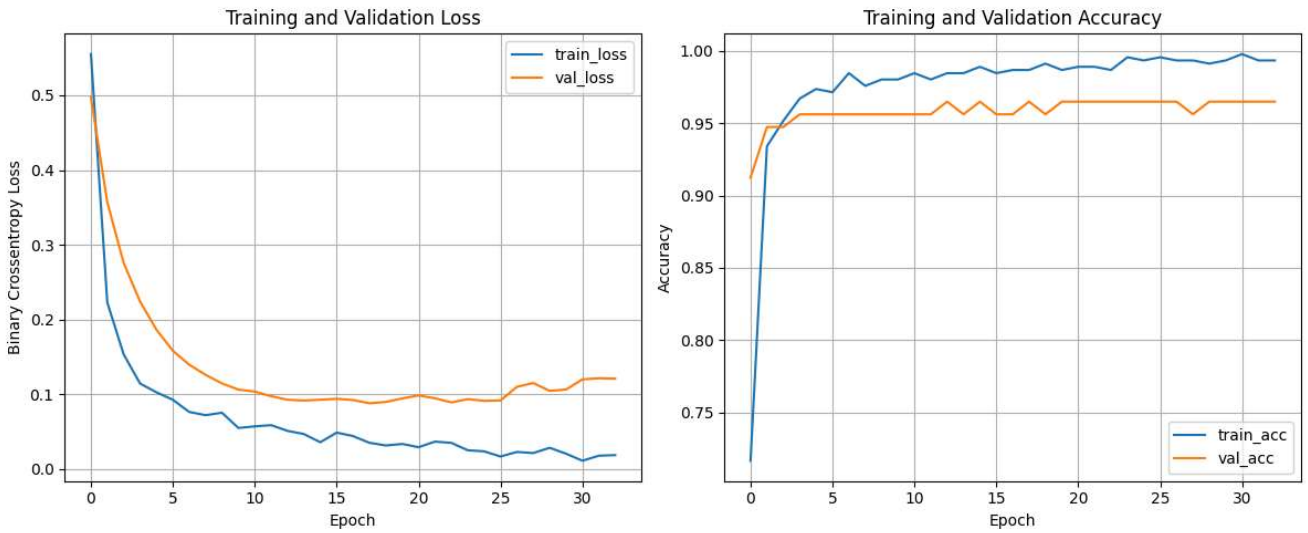
```
Epoch 14/200
15/15 - 0s - 8ms/step - accuracy: 0.9846 - loss: 0.0467 - val_accuracy: 0.9561 - val_loss: 0.0915
Epoch 15/200
15/15 - 0s - 8ms/step - accuracy: 0.9890 - loss: 0.0357 - val_accuracy: 0.9649 - val_loss: 0.0926
Epoch 16/200
15/15 - 0s - 8ms/step - accuracy: 0.9846 - loss: 0.0486 - val_accuracy: 0.9561 - val_loss: 0.0939
Epoch 17/200
15/15 - 0s - 9ms/step - accuracy: 0.9868 - loss: 0.0439 - val_accuracy: 0.9561 - val_loss: 0.0922
Epoch 18/200
15/15 - 0s - 9ms/step - accuracy: 0.9868 - loss: 0.0349 - val_accuracy: 0.9649 - val_loss: 0.0878
Epoch 19/200
15/15 - 0s - 8ms/step - accuracy: 0.9912 - loss: 0.0314 - val_accuracy: 0.9561 - val_loss: 0.0897
Epoch 20/200
15/15 - 0s - 8ms/step - accuracy: 0.9868 - loss: 0.0334 - val_accuracy: 0.9649 - val_loss: 0.0943
Epoch 21/200
15/15 - 0s - 8ms/step - accuracy: 0.9890 - loss: 0.0291 - val_accuracy: 0.9649 - val_loss: 0.0983
Epoch 22/200
15/15 - 0s - 8ms/step - accuracy: 0.9890 - loss: 0.0365 - val_accuracy: 0.9649 - val_loss: 0.0945
Epoch 23/200
15/15 - 0s - 8ms/step - accuracy: 0.9868 - loss: 0.0348 - val_accuracy: 0.9649 - val_loss: 0.0890
Epoch 24/200
15/15 - 0s - 11ms/step - accuracy: 0.9956 - loss: 0.0250 - val_accuracy: 0.9649 - val_loss: 0.0934
Epoch 25/200
15/15 - 0s - 21ms/step - accuracy: 0.9934 - loss: 0.0235 - val_accuracy: 0.9649 - val_loss: 0.0911
Epoch 26/200
15/15 - 0s - 13ms/step - accuracy: 0.9956 - loss: 0.0166 - val_accuracy: 0.9649 - val_loss: 0.0916
Epoch 27/200
15/15 - 0s - 12ms/step - accuracy: 0.9934 - loss: 0.0227 - val_accuracy: 0.9649 - val_loss: 0.1098
Epoch 28/200
15/15 - 0s - 13ms/step - accuracy: 0.9934 - loss: 0.0212 - val_accuracy: 0.9561 - val_loss: 0.1150
Epoch 29/200
15/15 - 0s - 13ms/step - accuracy: 0.9912 - loss: 0.0283 - val_accuracy: 0.9649 - val_loss: 0.1044
Epoch 30/200
15/15 - 0s - 11ms/step - accuracy: 0.9934 - loss: 0.0203 - val_accuracy: 0.9649 - val_loss: 0.1062
Epoch 31/200
15/15 - 0s - 13ms/step - accuracy: 0.9978 - loss: 0.0109 - val_accuracy: 0.9649 - val_loss: 0.1198
Epoch 32/200
15/15 - 0s - 12ms/step - accuracy: 0.9934 - loss: 0.0177 - val_accuracy: 0.9649 - val_loss: 0.1214
Epoch 33/200
15/15 - 0s - 13ms/step - accuracy: 0.9934 - loss: 0.0185 - val_accuracy: 0.9649 - val_loss: 0.1208
```

```
# Plot training & validation loss and accuracy
plt.figure(figsize=(12,5))

plt.subplot(1,2,1)
plt.plot(history.history['loss'], label='train_loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Binary Crossentropy Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.grid(True)

plt.subplot(1,2,2)
plt.plot(history.history['accuracy'], label='train_acc')
plt.plot(history.history['val_accuracy'], label='val_acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()
```



The training log for the Fully Connected Neural Network (FCNN) reveals three phases: rapid convergence, stabilization, and the onset of overfitting. Initially, the model showed powerful learning, with the training loss plummeting by 90% (from 0.5551 to 0.0548) within the first ten epochs, and validation accuracy quickly stabilizing at approximately 0.9561. The model achieved its optimal generalization and minimum validation loss (0.0878) around Epoch 18. Following this peak, the gap between the rapidly decreasing training loss and the plateaued/rising validation loss widened, indicating the beginning of overfitting. The Early Stopping mechanism successfully monitored this trend, correctly halting training after Epoch 33 (15 epochs of non-improvement) and restoring the model to the optimal weights achieved at Epoch 18, ensuring the final deployed model is robust and well-generalized with a validation accuracy of around 0.965.

```
# Evaluate Neural Network on validation set
y_val_proba_nn = model.predict(X_val).ravel()
y_val_pred_nn = (y_val_proba_nn >= 0.5).astype(int)

def compute_metrics(y_true, y_pred, name='Model', y_proba=None):
    acc = accuracy_score(y_true, y_pred)
    prec = precision_score(y_true, y_pred, zero_division=0)
    rec = recall_score(y_true, y_pred, zero_division=0)
    f1 = f1_score(y_true, y_pred, zero_division=0)
    print(f"--- {name} ---")
    print(f"Accuracy : {acc:.4f}")
    print(f"Precision: {prec:.4f}")
    print(f"Recall   : {rec:.4f}")
    print(f"F1 score : {f1:.4f}")
    if y_proba is not None:
        try:
            roc = roc_auc_score(y_true, y_proba)
            print(f"ROC AUC  : {roc:.4f}")
        except:
            pass
    print("Confusion matrix:")
    print(confusion_matrix(y_true, y_pred))
    print(classification_report(y_true, y_pred, zero_division=0))
    return {'accuracy': acc, 'precision': prec, 'recall': rec, 'f1': f1}

nn_metrics = compute_metrics(y_val, y_val_pred_nn, name='Neural Network', y_proba=y_val_proba_nn)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
--- Neural Network ---
Accuracy : 0.9649
Precision: 0.9722
Recall   : 0.9722
F1 score : 0.9722
ROC AUC  : 0.9944
Confusion matrix:
[[40  2]
 [ 2 70]]
              precision    recall  f1-score   support
```

```
              0       0.95      0.95      0.95        42
              1       0.97      0.97      0.97        72

       accuracy                           0.96       114
      macro avg       0.96      0.96      0.96       114
   weighted avg       0.96      0.96      0.96       114
```

```
# Logistic Regression baseline
logreg = LogisticRegression(max_iter=2000, random_state=seed)
logreg.fit(X_train, y_train)
y_lr_proba = logreg.predict_proba(X_val)[:,1]
y_lr_pred = logreg.predict(X_val)
lr_metrics = compute_metrics(y_val, y_lr_pred, name='Logistic Regression', y_proba=y_lr_proba)
```

```
--- Logistic Regression ---
Accuracy : 0.9825
Precision: 0.9861
Recall   : 0.9861
F1 score : 0.9861
ROC AUC  : 0.9954
Confusion matrix:
[[41  1]
 [ 1 71]]
              precision    recall  f1-score   support

           0       0.98      0.98      0.98        42
           1       0.99      0.99      0.99        72

    accuracy                           0.98       114
   macro avg       0.98      0.98      0.98       114
weighted avg       0.98      0.98      0.98       114
```

```
# Support Vector Machine baseline (RBF kernel)
# SVMs can be slow; probability=True enables predict_proba for ROC AUC
svc = SVC(kernel='rbf', probability=True, random_state=seed)
svc.fit(X_train, y_train)
y_svc_proba = svc.predict_proba(X_val)[:,1]
y_svc_pred = svc.predict(X_val)
svc_metrics = compute_metrics(y_val, y_svc_pred, name='SVM (RBF)', y_proba=y_svc_proba)
```

```
--- SVM (RBF) ---
Accuracy : 0.9825
Precision: 0.9861
Recall   : 0.9861
F1 score : 0.9861
ROC AUC  : 0.9950
Confusion matrix:
[[41  1]
 [ 1 71]]
              precision    recall  f1-score   support

           0       0.98      0.98      0.98        42
           1       0.99      0.99      0.99        72

    accuracy                           0.98       114
   macro avg       0.98      0.98      0.98       114
weighted avg       0.98      0.98      0.98       114
```

The performance of all three models on the 20% validation set is summarized below. The target positive class is Benign. All three models performed well on the Cancer dataset, with Logistic Regression and SVM having the same high accuracy levels for accuracy, precision, recall, and f1 score. The NueralNetwork had the worst performance out of the models, but still had high levels of accuracy, precision, recall, and f1 score

```
# Summary table for easy comparison
summary = pd.DataFrame([nn_metrics, lr_metrics, svc_metrics], index=['NeuralNet','Logistic','SVM'])
print("\nSummary comparison (validation set):")
display(summary)
```

Summary comparison (validation set):

	accuracy	precision	recall	f1
NeuralNet	0.964912	0.972222	0.972222	0.972222