Lab 9 result
Wonho Jeong
1002242697

Exercise 1
Code:

```
# Exercise 1: Train a DNN model and visualize training history

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping

# Load dataset
VarNames = ["signal", "l_1_pT", "l_1_eta","l_1_phi", "l_2_pT", "l_2_eta",
"l_2_phi",
            "MET", "MET_phi", "MET_rel", "axial_MET", "M_R", "M_TR_2",
"R", "MT2",
            "S_R", "M_Delta_R", "dPhi_r_b", "cos_theta_r1"]
df = pd.read_csv("SUSY.csv", names=VarNames)

# Train/test split
N_Max = 550000
N_Train = 500000
Train = df[:N_Train]
Test = df[N_Train:N_Max]

X_train = Train[VarNames[1:]].values
y_train = Train["signal"].values
X_test = Test[VarNames[1:]].values
y_test = Test["signal"].values

# Scale the data
scaler = StandardScaler()
```

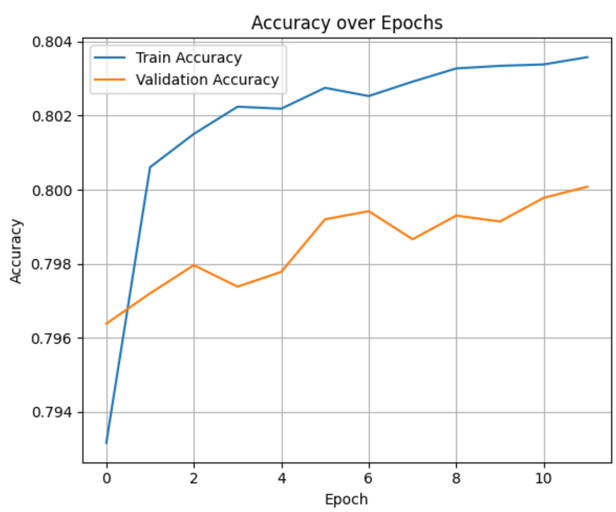
```
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Build the DNN model
def build_model(input_dim):
    model = Sequential([
        Dense(64, activation='relu', input_shape=(input_dim,)),
        Dense(32, activation='relu'),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer=Adam(0.001), loss='binary_crossentropy',
metrics=['accuracy'])
    return model

# Train the model
model = build_model(X_train.shape[1])
early_stop = EarlyStopping(monitor='val_loss', patience=3,
restore_best_weights=True)
history = model.fit(X_train, y_train, validation_data=(X_test, y_test),
                    epochs=20, batch_size=512, callbacks=[early_stop],
verbose=1)

# Plotting function
def plot_history(history, metric='accuracy'):
    plt.figure(figsize=(12, 5))

    # Accuracy
    plt.subplot(1, 2, 1)
    plt.plot(history.history[metric], label='Train Accuracy')
    plt.plot(history.history[f'val_{metric}'], label='Validation
Accuracy')
    plt.title('Accuracy over Epochs')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.grid(True)

    # Loss
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Train Loss')
```

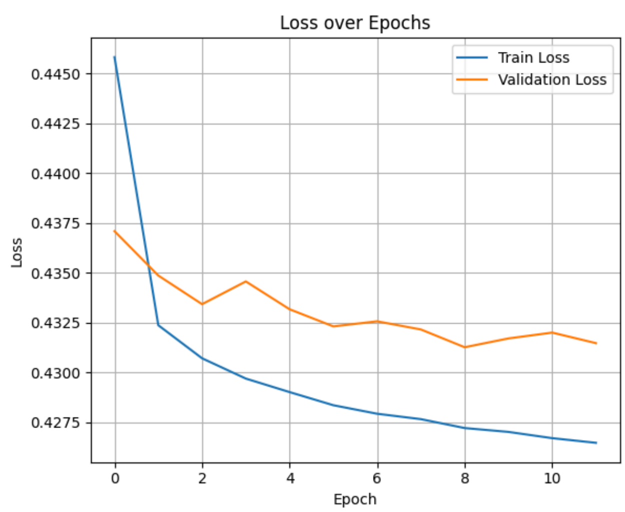
```
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title('Loss over Epochs')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True)

    plt.tight_layout()
    plt.show()

# Show results
plot_history(history)
```

Output:

/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
 super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/20
977/977 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.7792 - loss: 0.4691 - val_accuracy: 0.7964 - val_loss: 0.4371
Epoch 2/20
977/977 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.8004 - loss: 0.4331 - val_accuracy: 0.7972 - val_loss: 0.4349
Epoch 3/20
977/977 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.8019 - loss: 0.4308 - val_accuracy: 0.7980 - val_loss: 0.4334
Epoch 4/20
977/977 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.8026 - loss: 0.4287 - val_accuracy: 0.7974 - val_loss: 0.4346
Epoch 5/20
977/977 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.8023 - loss: 0.4290 - val_accuracy: 0.7978 - val_loss: 0.4332
Epoch 6/20
977/977 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.8029 - loss: 0.4281 - val_accuracy: 0.7992 - val_loss: 0.4323
Epoch 7/20
977/977 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8026 - loss: 0.4283 - val_accuracy: 0.7994 - val_loss: 0.4326
Epoch 8/20

977/977 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.8029 - loss: 0.4274 - val_accuracy: 0.7987 - val_loss: 0.4322
Epoch 9/20
977/977 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.8027 - loss: 0.4277 - val_accuracy: 0.7993 - val_loss: 0.4313
Epoch 10/20
977/977 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.8037 - loss: 0.4268 - val_accuracy: 0.7991 - val_loss: 0.4317
Epoch 11/20
977/977 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.8029 - loss: 0.4271 - val_accuracy: 0.7998 - val_loss: 0.4320
Epoch 12/20
977/977 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.8039 - loss: 0.4258 - val_accuracy: 0.8001 - val_loss: 0.4315





Exercise 2

Code:

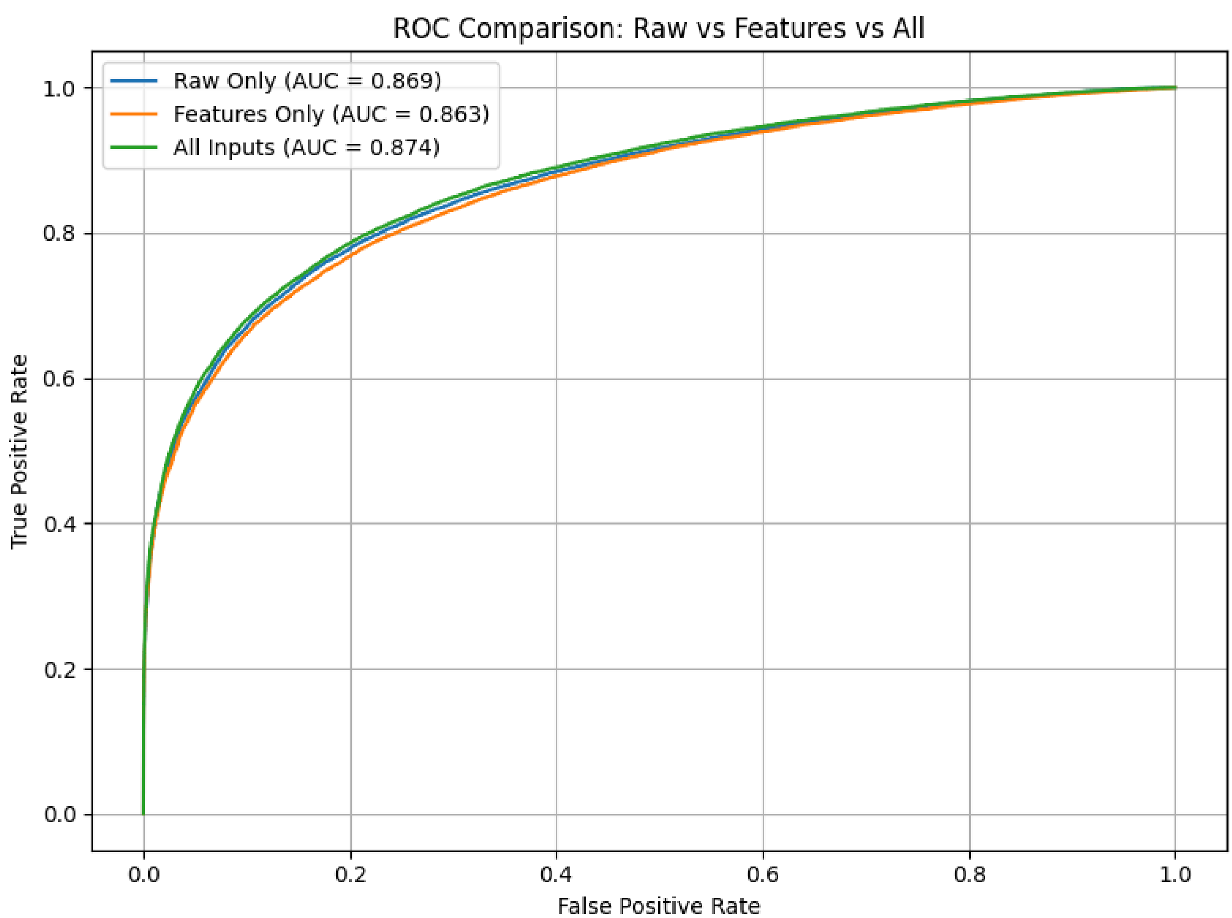
```
# Exercise 2 - DNN performance comparison: Raw vs Features vs Raw+Features

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import roc_curve, auc
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
```

```
# 1. Load the SUSY dataset
VarNames = ["signal", "l_1_pT", "l_1_eta", "l_1_phi", "l_2_pT", "l_2_eta",
"l_2_phi",
            "MET", "MET_phi", "MET_rel", "axial_MET", "M_R", "M_TR_2",
"R", "MT2",
            "S_R", "M_Delta_R", "dPhi_r_b", "cos_theta_r1"]
df = pd.read_csv("SUSY.csv", names=VarNames)

# 2. Split data into training and testing sets
N_Max = 550000
N_Train = 500000
Train = df[:N_Train]
Test = df[N_Train:N_Max]

# 3. Define input variable groups
RawNames = ["l_1_pT", "l_1_eta", "l_1_phi", "l_2_pT", "l_2_eta",
"l_2_phi", "MET", "MET_phi"]
FeatureNames = ['S_R', 'MET_rel', 'M_R', 'dPhi_r_b', 'M_Delta_R',
                'MT2', 'axial_MET', 'R', 'M_TR_2', 'cos_theta_r1']
AllNames = RawNames + FeatureNames

# 4. Standardize input features
scaler = StandardScaler()
X_raw_train = scaler.fit_transform(Train[RawNames])
X_raw_test = scaler.transform(Test[RawNames])
X_feat_train = scaler.fit_transform(Train[FeatureNames])
X_feat_test = scaler.transform(Test[FeatureNames])
X_all_train = scaler.fit_transform(Train[AllNames])
X_all_test = scaler.transform(Test[AllNames])

y_train = Train["signal"].values
y_test = Test["signal"].values

# 5. DNN training function
def train_dnn(X_train, X_test, y_train, y_test, input_dim):
    model = Sequential([
        Dense(64, activation='relu', input_shape=(input_dim,)),
        Dense(32, activation='relu'),
        Dense(1, activation='sigmoid')
    ])
```

```
    model.compile(optimizer=Adam(learning_rate=0.001),
                  loss='binary_crossentropy', metrics=['accuracy'])
    model.fit(X_train, y_train, epochs=10, batch_size=512,
              validation_data=(X_test, y_test), verbose=0)
    return model

# 6. Train 3 DNN models
model_raw = train_dnn(X_raw_train, X_raw_test, y_train, y_test,
len(RawNames))
model_feat = train_dnn(X_feat_train, X_feat_test, y_train, y_test,
len(FeatureNames))
model_all = train_dnn(X_all_train, X_all_test, y_train, y_test,
len(AllNames))

# 7. Compute ROC and AUC
def compute_roc_auc(model, X_test, y_test):
    y_pred = model.predict(X_test).ravel()
    fpr, tpr, _ = roc_curve(y_test, y_pred)
    auc_score = auc(fpr, tpr)
    return fpr, tpr, auc_score

fpr_raw, tpr_raw, auc_raw = compute_roc_auc(model_raw, X_raw_test, y_test)
fpr_feat, tpr_feat, auc_feat = compute_roc_auc(model_feat, X_feat_test,
y_test)
fpr_all, tpr_all, auc_all = compute_roc_auc(model_all, X_all_test, y_test)

# 8. Plot ROC curves for all models
plt.figure(figsize=(8, 6))
plt.plot(fpr_raw, tpr_raw, label=f"Raw Only (AUC = {auc_raw:.3f})")
plt.plot(fpr_feat, tpr_feat, label=f"Features Only (AUC =
{auc_feat:.3f})")
plt.plot(fpr_all, tpr_all, label=f"All Inputs (AUC = {auc_all:.3f})")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Comparison: Raw vs Features vs All")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

Output:

/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step



Exercise 3
Code:

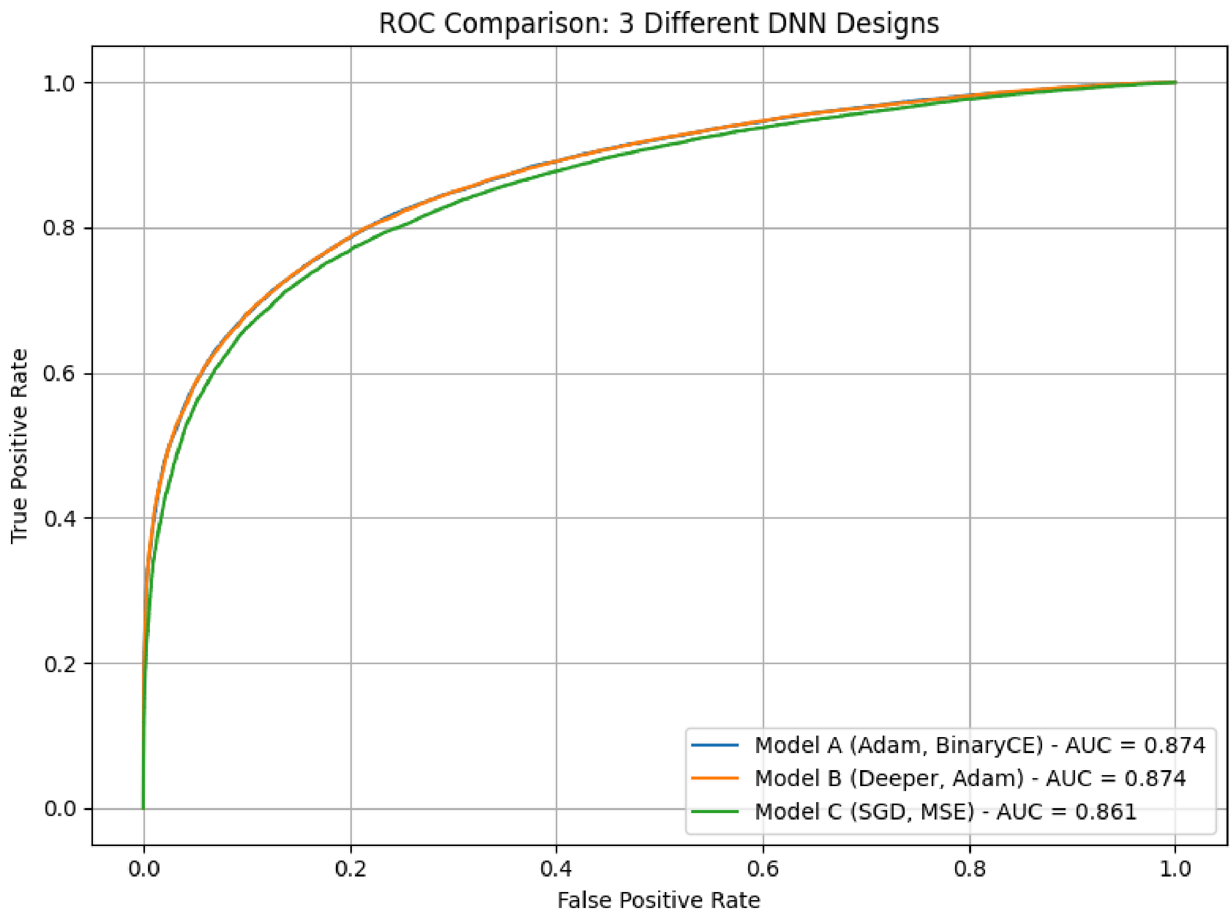
```
# Exercise 3: Compare 3 different DNN architectures

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import roc_curve, auc
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam, SGD
from tensorflow.keras.losses import BinaryCrossentropy, MeanSquaredError

# Load dataset
VarNames = ["signal", "l_1_pT", "l_1_eta", "l_1_phi", "l_2_pT", "l_2_eta",
"l_2_phi",
            "MET", "MET_phi", "MET_rel", "axial_MET", "M_R", "M_TR_2",
"R", "MT2",
            "S_R", "M_Delta_R", "dPhi_r_b", "cos_theta_r1"]
df = pd.read_csv("SUSY.csv", names=VarNames)

# Data split
N_Max = 550000
N_Train = 500000
Train = df[:N_Train]
Test = df[N_Train:N_Max]

# Use all variables
AllNames = VarNames[1:]

scaler = StandardScaler()
X_train = scaler.fit_transform(Train[AllNames])
X_test = scaler.transform(Test[AllNames])
y_train = Train["signal"].values
y_test = Test["signal"].values

# Model A: Basic architecture, Adam, binary crossentropy
def build_model_A(input_dim):
```

```
    model = Sequential([
        Dense(64, activation='relu', input_shape=(input_dim,)),
        Dense(32, activation='relu'),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer=Adam(0.001), loss=BinaryCrossentropy(),
metrics=['accuracy'])
    return model

# Model B: Deeper architecture
def build_model_B(input_dim):
    model = Sequential([
        Dense(128, activation='relu', input_shape=(input_dim,)),
        Dense(64, activation='relu'),
        Dense(32, activation='relu'),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer=Adam(0.001), loss=BinaryCrossentropy(),
metrics=['accuracy'])
    return model

# Model C: Different optimizer and loss function
def build_model_C(input_dim):
    model = Sequential([
        Dense(64, activation='relu', input_shape=(input_dim,)),
        Dense(64, activation='relu'),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer=SGD(learning_rate=0.01),
loss=MeanSquaredError(), metrics=['accuracy'])
    return model

# Train all models
model_A = build_model_A(X_train.shape[1])
model_B = build_model_B(X_train.shape[1])
model_C = build_model_C(X_train.shape[1])

model_A.fit(X_train, y_train, epochs=10, batch_size=512, verbose=0)
model_B.fit(X_train, y_train, epochs=10, batch_size=512, verbose=0)
model_C.fit(X_train, y_train, epochs=10, batch_size=512, verbose=0)
```

```python
# Function to evaluate
def compute_roc_auc(model, X_test, y_test):
    y_pred = model.predict(X_test).ravel()
    fpr, tpr, _ = roc_curve(y_test, y_pred)
    auc_score = auc(fpr, tpr)
    return fpr, tpr, auc_score

# Evaluate each model
fpr_A, tpr_A, auc_A = compute_roc_auc(model_A, X_test, y_test)
fpr_B, tpr_B, auc_B = compute_roc_auc(model_B, X_test, y_test)
fpr_C, tpr_C, auc_C = compute_roc_auc(model_C, X_test, y_test)

# Plot
plt.figure(figsize=(8, 6))
plt.plot(fpr_A, tpr_A, label=f"Model A (Adam, BinaryCE) - AUC =
{auc_A:.3f}")
plt.plot(fpr_B, tpr_B, label=f"Model B (Deeper, Adam) - AUC =
{auc_B:.3f}")
plt.plot(fpr_C, tpr_C, label=f"Model C (SGD, MSE) - AUC = {auc_C:.3f}")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Comparison: 3 Different DNN Designs")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

Output:

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step
```



Exercise 4

Code:

```
# Exercise 4: Compare Lab 8 models + best DNN model from Exercise 3

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import roc_curve, auc
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.naive_bayes import GaussianNB
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam

# 1. Load data
```

```
VarNames = ["signal", "l_1_pT", "l_1_eta", "l_1_phi", "l_2_pT", "l_2_eta",
"l_2_phi",
            "MET", "MET_phi", "MET_rel", "axial_MET", "M_R", "M_TR_2",
"R", "MT2",
            "S_R", "M_Delta_R", "dPhi_r_b", "cos_theta_r1"]
df = pd.read_csv("SUSY.csv", names=VarNames)

# 2. Prepare train/test split
N_Max = 550000
N_Train = 500000
Train = df[:N_Train]
Test = df[N_Train:N_Max]

AllNames = VarNames[1:]
scaler = StandardScaler()
X_train = scaler.fit_transform(Train[AllNames])
X_test = scaler.transform(Test[AllNames])
y_train = Train["signal"].values
y_test = Test["signal"].values

# 3. Best DNN model from Exercise 3 (Model A)
def build_model_A(input_dim):
    model = Sequential([
        Dense(64, activation='relu', input_shape=(input_dim,)),
        Dense(32, activation='relu'),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer=Adam(0.001), loss='binary_crossentropy',
metrics=['accuracy'])
    return model

model_A = build_model_A(X_train.shape[1])
model_A.fit(X_train, y_train, epochs=10, batch_size=512, verbose=0)

# 4. Scikit-learn classifiers from Lab 8
models = [
    (LogisticRegression(max_iter=1000), "Logistic Regression"),
    (SGDClassifier(loss='log_loss', max_iter=1000, random_state=42), "SGD
Classifier"),
    (GaussianNB(), "Naive Bayes")
```

```python
]

def evaluate_classifier(model, X_train, X_test, y_train, y_test,
label=""):
    model.fit(X_train, y_train)
    if hasattr(model, "predict_proba"):
        y_score = model.predict_proba(X_test)[:, 1]
    else:
        y_score = model.decision_function(X_test)
    fpr, tpr, _ = roc_curve(y_test, y_score)
    auc_score = auc(fpr, tpr)
    return fpr, tpr, auc_score, label

# 5. Evaluate traditional classifiers
results = []
for clf, label in models:
    fpr, tpr, auc_score, name = evaluate_classifier(clf, X_train, X_test,
y_train, y_test, label)
    results.append((fpr, tpr, auc_score, name))

# 6. Evaluate DNN
y_score_dnn = model_A.predict(X_test).ravel()
fpr_dnn, tpr_dnn, _ = roc_curve(y_test, y_score_dnn)
auc_dnn = auc(fpr_dnn, tpr_dnn)
results.append((fpr_dnn, tpr_dnn, auc_dnn, "Best DNN (Model A)"))

# 7. Plot ROC comparison
plt.figure(figsize=(8, 6))
for fpr, tpr, auc_score, label in results:
    plt.plot(fpr, tpr, label=f"{label} (AUC = {auc_score:.3f})")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Comparison: Lab 8 Models + DNN")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

Output:

/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
super().__init__(activity_regularizer=activity_regularizer, **kwargs)
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step