Nama : Khalishah

NIM : 1103213045

Laporan MLP Classification Machine Learning Week 10

Import library yg butuhkan:

Import libraries

```
[ ]  Generated code may be subject to a license | UncleThree0402/PyTorch_FFN_HeartDisease | LPapakostas/welding_temp···
import pandas as pd
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Memuat dataset dan menampilkan 5 baris dataset:

Load dataset

```
[ ] df = pd.read_csv('/content/heart_failure_clinical_records_dataset.csv')
df.head()
```

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_
0	75.0	0	582	0	20	
1	55.0	0	7861	0	38	
2	65.0	0	146	0	20	
3	50.0	1	111	0	20	
4	65.0	1	160	1	20	

Next steps: Generate code with df | View recommended plots | New interactive sheet

Menampilkan informasi dataset:

```
[ ] df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 299 entries, 0 to 298
Data columns (total 13 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   age                       299 non-null    float64
 1   anaemia                   299 non-null    int64  
 2   creatinine_phosphokinase  299 non-null    int64  
 3   diabetes                  299 non-null    int64  
 4   ejection_fraction         299 non-null    int64  
 5   high_blood_pressure       299 non-null    int64  
 6   platelets                 299 non-null    float64
 7   serum_creatinine          299 non-null    float64
 8   serum_sodium              299 non-null    int64  
 9   sex                       299 non-null    int64  
 10  smoking                   299 non-null    int64  
 11  time                      299 non-null    int64  
 12  DEATH_EVENT               299 non-null    int64  
dtypes: float64(3), int64(10)
memory usage: 30.5 KB
```

Memberikan ringkasan statistik dari kolom numerik dalam DataFrame, seperti rata-rata, standar deviasi, dan nilai minimum/maksimum:

```
[ ] df.describe()
```

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fractio
count	299.000000	299.000000	299.000000	299.000000	299.00000
mean	60.833893	0.431438	581.839465	0.418060	38.08361
std	11.894809	0.496107	970.287881	0.494067	11.83484
min	40.000000	0.000000	23.000000	0.000000	14.00000
25%	51.000000	0.000000	116.500000	0.000000	30.00000
50%	60.000000	0.000000	250.000000	0.000000	38.00000
75%	70.000000	1.000000	582.000000	1.000000	45.00000
max	95.000000	1.000000	7861.000000	1.000000	80.00000

Menghitung jumlah nilai NaN(Not a Number) atau data yang hilang di setiap kolom dalam DataFrame:

```
[ ] df.isna().sum()
```

	0
age	0
anaemia	0
creatinine_phosphokinase	0
diabetes	0
ejection_fraction	0
high_blood_pressure	0
platelets	0
serum_creatinine	0
serum_sodium	0
sex	0
smoking	0
time	0
DEATH_EVENT	0

dtype: int64

Menampilkan kolom pada dataset:

```
[ ] df.columns
```

```
Index(['age', 'anaemia', 'creatinine_phosphokinase', 'diabetes',
       'ejection_fraction', 'high_blood_pressure', 'platelets',
       'serum_creatinine', 'serum_sodium', 'sex', 'smoking', 'time',
       'DEATH_EVENT'],
      dtype='object')
```

Mengonversi kolom-kolom dalam yang berisi kategori:

Encoding categorical columns using LabelEncoder

```
[ ] label_cols = ['anaemia', 'diabetes', 'high_blood_pressure', 'sex', 'smoking', 'DEATH_EVENT']
    label_encoder = LabelEncoder()

    for col in label_cols:
        df[col] = label_encoder.fit_transform(df[col])
```

Memisahkan fitur (X) dan target (y) dalam dataset:

Determine features (X) and targets (y)

```
[ ]  X = df.drop(['DEATH_EVENT'], axis=1)
     y = df['DEATH_EVENT']
```

```
[ ]  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Melakukan standaeisasi fitur agar memiliki skala yang sama (mean = 0, standar deviasi = 1):

Standardization of numeric features

```
[ ]  scaler = StandardScaler()
     X_train_scaled = scaler.fit_transform(X_train)
     X_test_scaled = scaler.transform(X_test)
```

Mengonversi data latih dan uji menjadi tensor PyTorch, yang diperlukan saat bekerja dengan model pembelajaran mesin di PyTorch:

Convert data to tensor

```
[ ]  X_train_tensor = torch.tensor(X_train_scaled, dtype=torch.float32)
     y_train_tensor = torch.tensor(y_train.values, dtype=torch.long)
     X_test_tensor = torch.tensor(X_test_scaled, dtype=torch.float32)
     y_test_tensor = torch.tensor(y_test.values, dtype=torch.long)
```

Mendefinisikan model Multi-Layer Perceptron (MLP) untuk tugas klasifikasi dengan lapisan tersembunyi yang dapat disesuaikan. Model ini menerima data input dengan ukuran yang ditentukan, lalu melewatkan data melalui beberapa lapisan tersembunyi yang memiliki jumlah neuron yang dapat diatur, serta fungsi aktivasi yang dipilih (seperti ReLU atau Sigmoid). Output model ini adalah hasil klasifikasi, yaitu jumlah neuron pada lapisan output sesuai dengan jumlah kelas target (misalnya, dua kelas untuk klasifikasi biner). Model ini fleksibel, memungkinkan eksperimen dengan berbagai konfigurasi lapisan dan fungsi aktivasi untuk mencari konfigurasi yang terbaik dalam memprediksi kelas target.

Constructing an MLP model for regression

```
[ ]  # Defining the MLP model for classification
     class MLPClassification(nn.Module):
         def __init__(self, input_size, hidden_layers, neurons, activation):
             super(MLPClassification, self).__init__()
             self.input_size = input_size
             self.hidden_layers = hidden_layers
             self.neurons = neurons
             self.activation = activation

             # Creating input to hidden layer
             layers = []
             layers.append(nn.Linear(self.input_size, self.neurons))

             # Adding hidden layers
             for _ in range(self.hidden_layers - 1):
                 layers.append(self.activation())  # Activation function
                 layers.append(nn.Linear(self.neurons, self.neurons))

             # Adding the output layer with the number of classes (e.g., 2 or more)
             layers.append(nn.Linear(self.neurons, len(y.unique())))
             self.model = nn.Sequential(*layers)

         def forward(self, x):
             # Forward pass through the model
             return self.model(x)
```

Menentukan perangkat yang akan digunakan untuk pelatihan model, baik GPU (CUDA) atau CPU:

Device setup (GPU if available)

```
[ ]  device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

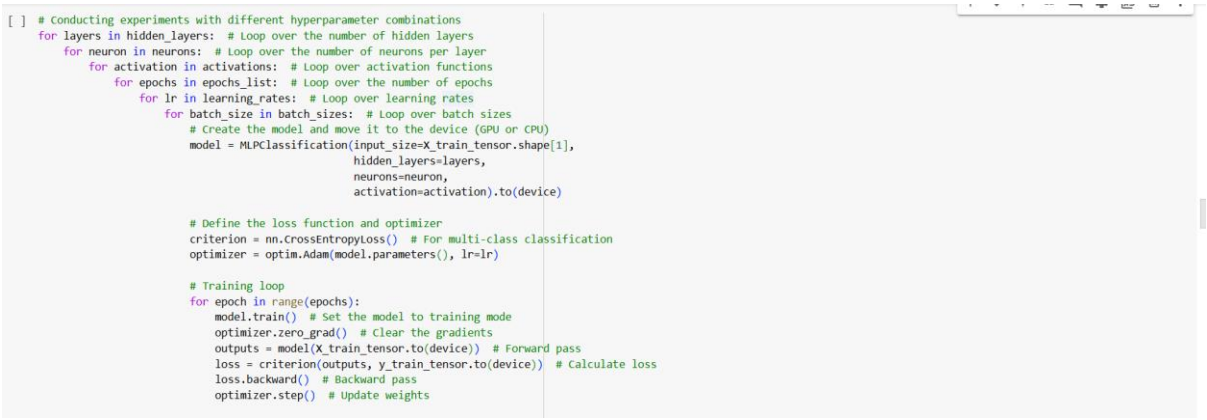
Mendefinisikan beberapa hyperparameters untuk eksperimen pelatihan model MLP (Multi-Layer Perceptron):

Hyperparameters to be tested

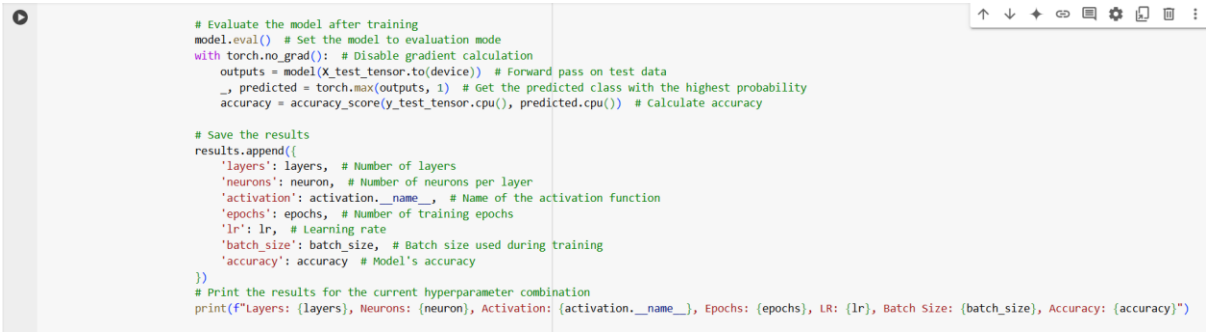
```
     hidden_layers = [1, 2, 3]
     neurons = [4, 8, 16, 32, 64]
     activations = [nn.Sigmoid, nn.Softmax, nn.ReLU, nn.Tanh]
     epochs_list = [1,10,25,50,100,250]
     learning_rates = [10, 1, 0.1, 0.01, 0.001, 0.0001]
     batch_sizes = [16,32,64,128,256,512]

     results = []
```

Kode ini melakukan eksperimen dengan berbagai kombinasi hyperparameter untuk melatih model MLP (Multi-Layer Perceptron) dalam tugas klasifikasi. Hyperparameter yang diuji meliputi jumlah lapisan tersembunyi, jumlah neuron per lapisan, fungsi aktivasi, jumlah epoch, learning

rate, dan ukuran batch. Untuk setiap kombinasi hyperparameter, model dilatih menggunakan CrossEntropyLoss dan Adam optimizer. Setelah pelatihan, model diuji pada data uji dan akurasi dihitung. Hasil eksperimen (termasuk nilai hyperparameter dan akurasi) disimpan dalam sebuah daftar dan dicetak setelah setiap iterasi. Tujuan eksperimen ini adalah untuk menemukan kombinasi hyperparameter terbaik yang memberikan kinerja model optimal.



```
[ ] # Conducting experiments with different hyperparameter combinations
    for layers in hidden_layers:  # Loop over the number of hidden layers
        for neuron in neurons:  # Loop over the number of neurons per layer
            for activation in activations:  # Loop over activation functions
                for epochs in epochs_list:  # Loop over the number of epochs
                    for lr in learning_rates:  # Loop over learning rates
                        for batch_size in batch_sizes:  # Loop over batch sizes
                            # Create the model and move it to the device (GPU or CPU)
                            model = MLPClassification(input_size=X_train_tensor.shape[1],
                                                      hidden_layers=layers,
                                                      neurons=neuron,
                                                      activation=activation).to(device)

                            # Define the loss function and optimizer
                            criterion = nn.CrossEntropyLoss()  # For multi-class classification
                            optimizer = optim.Adam(model.parameters(), lr=lr)

                            # Training loop
                            for epoch in range(epochs):
                                model.train()  # Set the model to training mode
                                optimizer.zero_grad()  # Clear the gradients
                                outputs = model(X_train_tensor.to(device))  # Forward pass
                                loss = criterion(outputs, y_train_tensor.to(device))  # Calculate loss
                                loss.backward()  # Backward pass
                                optimizer.step()  # Update weights
```



```
                            # Evaluate the model after training
                            model.eval()  # Set the model to evaluation mode
                            with torch.no_grad():  # Disable gradient calculation
                                outputs = model(X_test_tensor.to(device))  # Forward pass on test data
                                _, predicted = torch.max(outputs, 1)  # Get the predicted class with the highest probability
                                accuracy = accuracy_score(y_test_tensor.cpu(), predicted.cpu())  # Calculate accuracy

                            # Save the results
                            results.append({
                                'layers': layers,  # Number of layers
                                'neurons': neuron,  # Number of neurons per layer
                                'activation': activation.__name__,  # Name of the activation function
                                'epochs': epochs,  # Number of training epochs
                                'lr': lr,  # Learning rate
                                'batch_size': batch_size,  # Batch size used during training
                                'accuracy': accuracy  # Model's accuracy
                            })
                            # Print the results for the current hyperparameter combination
                            print(f"Layers: {layers}, Neurons: {neuron}, Activation: {activation.__name__}, Epochs: {epochs}, LR: {lr}, Batch Size: {batch_size}, Accuracy: {accuracy}")
```

Menyimpan hasil eksperimen dalam sebuah DataFrame pandas dan kemudian mengekspor data tersebut ke dalam file CSV:

Convert the results to a DataFrame and save them to CSV.

```
[ ] results_df = pd.DataFrame(results)
    results_df.to_csv("mlp_classification_hidden layer 123.csv", index=False)
    print("All results have been saved to 'mlp_classification_hidden layer 123.csv'.")
```

```
All results have been saved to 'mlp_classification_hidden layer 123.csv'.
```

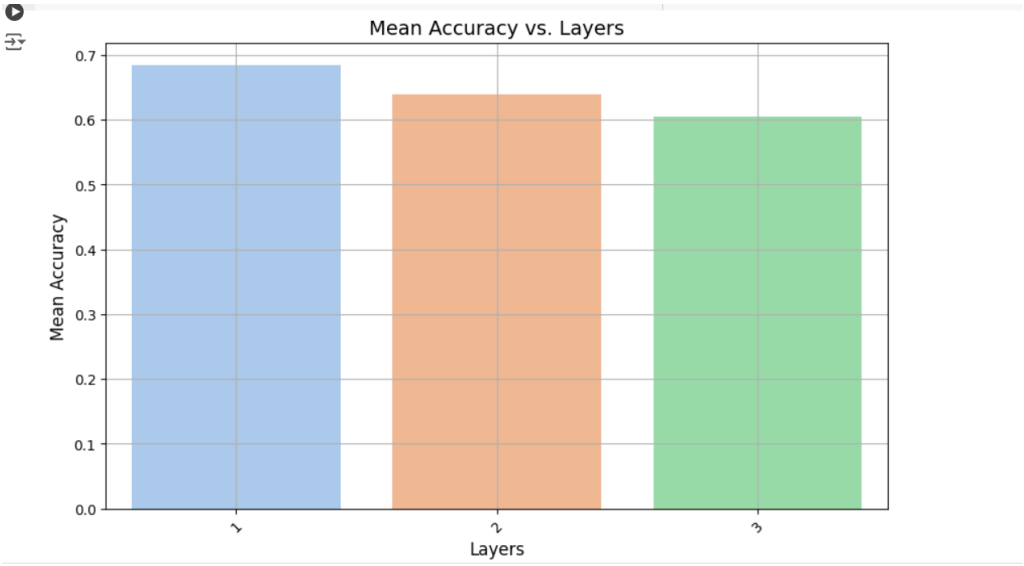
Kode ini digunakan untuk memvisualisasikan hubungan antara berbagai hyperparameter model MLP Classification dan rata-rata akurasi yang dihasilkan selama eksperimen. Data hasil eksperimen dimuat dari file CSV, kemudian akurasi rata-rata dihitung untuk setiap kombinasi hyperparameter menggunakan .groupby(). Selanjutnya, bar plot dibuat untuk setiap hyperparameter seperti jumlah lapisan, jumlah neuron, fungsi aktivasi, jumlah epoch, learning rate, dan ukuran batch. Visualisasi ini membantu mengidentifikasi bagaimana setiap hyperparameter memengaruhi performa model, memungkinkan analisis yang lebih mudah untuk menemukan kombinasi optimal.

```python
import matplotlib.pyplot as plt
import seaborn as sns

# Load the DataFrame containing the experiment results
# Ensure 'results_df' exists after executing the previous code
results_df = pd.read_csv("mlp_classification_hidden layer 123.csv")

# Calculate the mean accuracy for each hyperparameter combination
# Changed 'mae' to 'accuracy' to compute and visualize accuracy
mean_accuracy_by_hyperparameter = results_df.groupby(['layers', 'neurons', 'activation', 'epochs', 'lr', 'batch_size'])['accuracy'].mean().reset_index()

# List of hyperparameters to visualize
hyperparameters = ['layers', 'neurons', 'activation', 'epochs', 'lr', 'batch_size']

# Create bar plots for each hyperparameter
for param in hyperparameters:
    plt.figure(figsize=(10, 6))
    # Changed 'mae' to 'accuracy' in sns.barplot and ylabel
    sns.barplot(data=mean_accuracy_by_hyperparameter, x=param, y='accuracy', ci=None, palette="pastel")
    plt.title(f'Mean Accuracy vs. {param.capitalize()}', fontsize=14)  # Changed title to represent Accuracy
    plt.xlabel(param.capitalize(), fontsize=12)
    plt.ylabel('Mean Accuracy', fontsize=12)  # Changed ylabel to represent Accuracy
    plt.xticks(rotation=45)
    plt.grid(True)
    plt.show()
```

Mean Accuracy vs. Layers: Diagram ini menunjukkan bagaimana perubahan jumlah layers (lapisan) dalam model neural network memengaruhi akurasi rata-rata.

- Sumbu x: Jumlah layer (contoh: 1, 2, 3, dst.).
- Sumbu y: Akurasi rata-rata.

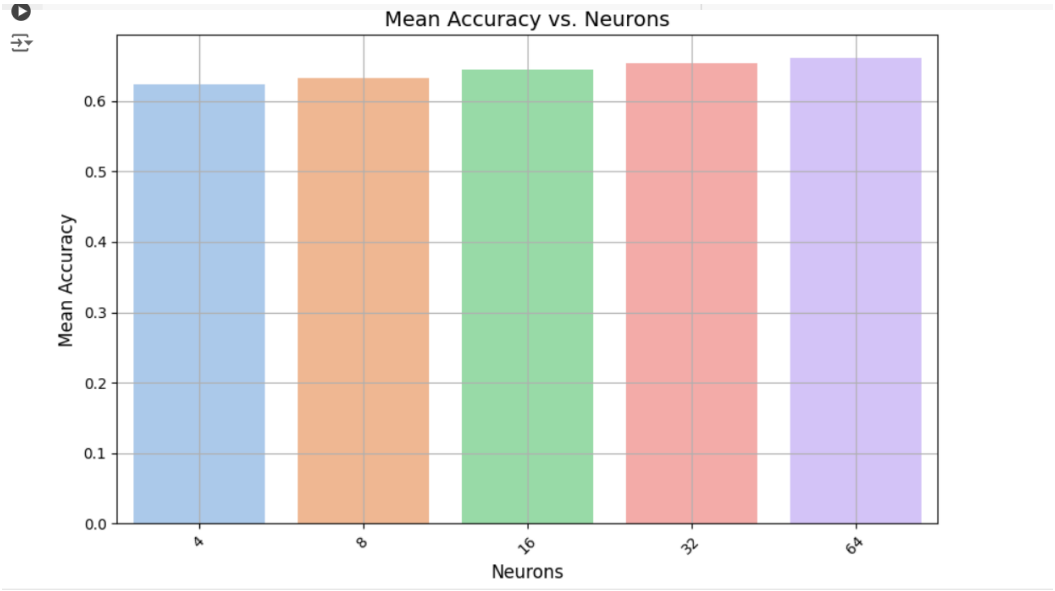
Kesimpulan: Akurasi tidak terlalu berubah secara signifikan dengan bertambahnya jumlah layer.



Mean Accuracy vs. Neurons: Diagram ini menunjukkan hubungan antara jumlah neuron per layer dengan akurasi rata-rata.

- Sumbu x: Jumlah neuron (contoh: 8, 16, 32, dst.).
- Sumbu y: Akurasi rata-rata.

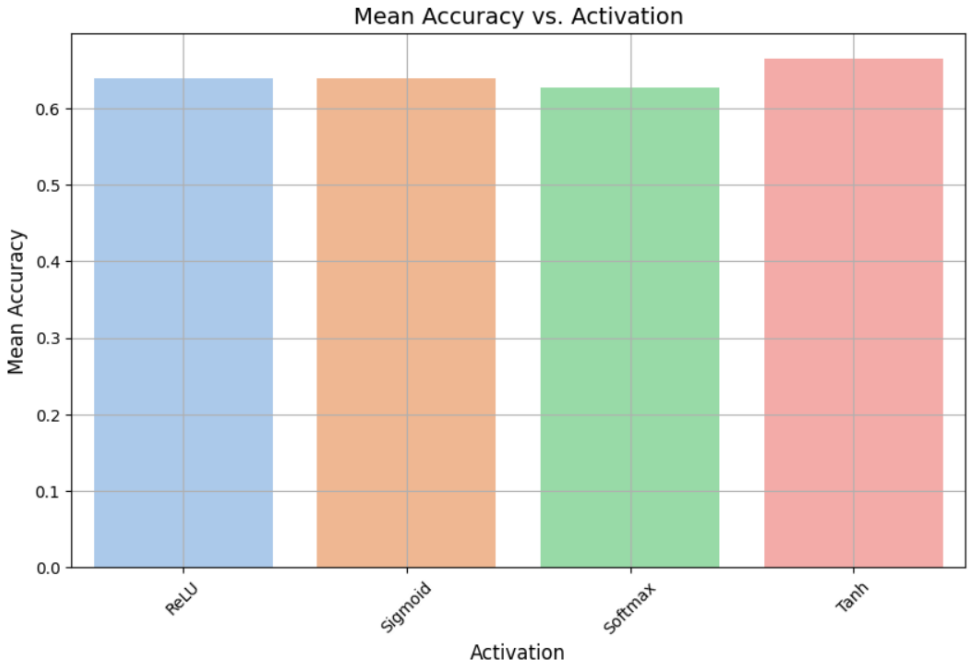
Kesimpulan: Akurasi cenderung stabil meskipun jumlah neuron berubah.



Mean Accuracy vs. Activation: Diagram ini membandingkan akurasi rata-rata untuk berbagai jenis fungsi aktivasi.

- Sumbu x: Fungsi aktivasi (contoh: ReLU, Sigmoid, Softmax, Tanh).
- Sumbu y: Akurasi rata-rata.

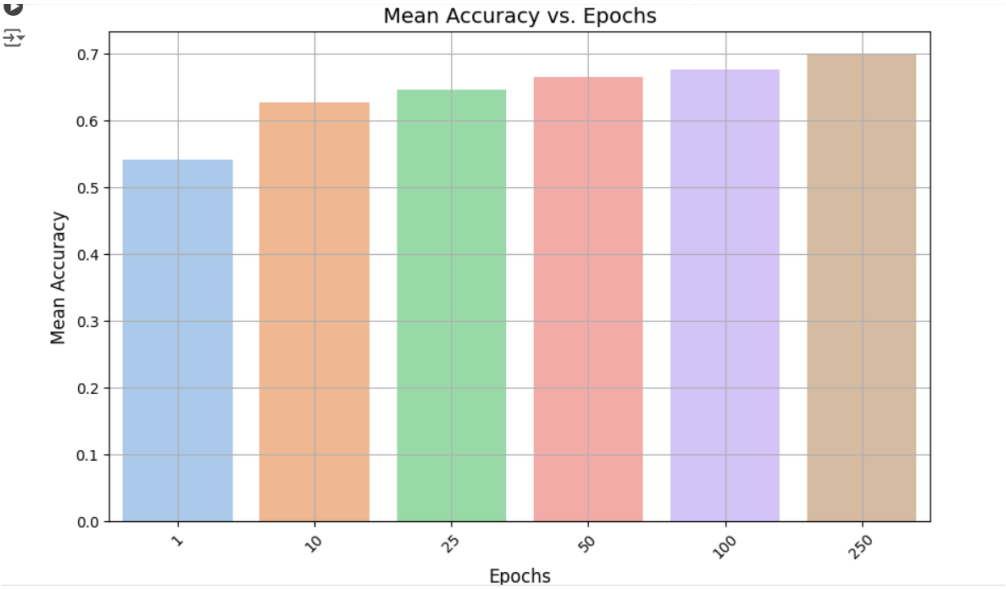
Kesimpulan: Akurasi rata-rata cukup seragam untuk berbagai fungsi aktivasi, dengan sedikit perbedaan.



Mean Accuracy vs. Epochs: Diagram ini menunjukkan pengaruh jumlah epochs (jumlah iterasi pelatihan) terhadap akurasi rata-rata.

- Sumbu x: Jumlah epoch (contoh: 1, 10, 25, dst.).
- Sumbu y: Akurasi rata-rata.

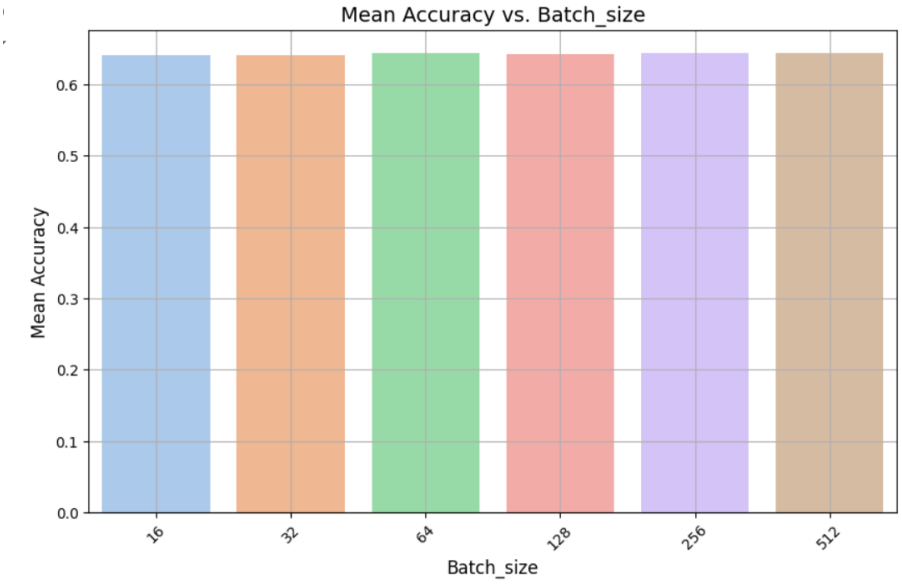
Kesimpulan: Akurasi meningkat dengan bertambahnya jumlah epoch hingga mencapai titik tertentu, kemudian stabil.



Mean Accuracy vs. Batch Size: Diagram ini mengevaluasi efek dari ukuran batch yang digunakan selama pelatihan terhadap akurasi rata-rata.

- Sumbu x: Ukuran batch (contoh: 16, 32, 64, dst.).
- Sumbu y: Akurasi rata-rata.

Kesimpulan: Ukuran batch tidak memberikan pengaruh signifikan terhadap akurasi rata-rata.



Kode ini membuat bar plot untuk memvisualisasikan akurasi model MLP Classification berdasarkan fungsi aktivasi dan jumlah neuron. Data hasil eksperimen dimuat dari file CSV, kemudian akurasi (accuracy) diplot di sumbu y dengan fungsi aktivasi (activation) di sumbu x. Jumlah neuron divisualisasikan sebagai kategori berbeda menggunakan warna pada batang (dengan parameter hue='neurons'). Judul, label sumbu, dan legenda ditambahkan untuk mempermudah interpretasi. Plot ini membantu melihat pola hubungan antara fungsi aktivasi, jumlah neuron, dan akurasi model, mempermudah analisis kombinasi hyperparameter terbaik.

```python
# Load the DataFrame containing the experiment results
# Ensure 'results_df' exists after executing the previous code
results_df = pd.read_csv("mlp_classification_hidden layer 123.csv")

# Plot accuracy for each combination of hyperparameters
plt.figure(figsize=(10, 6))

# Change 'mse' to 'accuracy' for the y-axis parameter
sns.barplot(data=results_df, x='activation', y='accuracy', hue='neurons', palette='pastel')

# Add title and labels
plt.title('Accuracy for Various Hyperparameter Combinations', fontsize=16)  # Changed the title to represent Accuracy
plt.xlabel('Activation Function', fontsize=12)
plt.ylabel('Accuracy', fontsize=12)  # Changed the y-axis label to represent Accuracy
plt.legend(title='Number of Neurons', title_fontsize='13', loc='upper right')

# Display the plot with proper layout
plt.tight_layout()
plt.show()
```

Diagram dibawah menggambarkan hubungan antara fungsi aktivasi dan jumlah neuron dengan akurasi model machine learning. Sumbu horizontal menampilkan empat jenis fungsi aktivasi yang digunakan dalam neural network, yaitu Sigmoid, Softmax, ReLU, dan Tanh, sementara sumbu vertikal menunjukkan nilai akurasi yang dicapai. Setiap warna pada batang menggambarkan jumlah neuron yang berbeda dalam hidden layer, yaitu 4, 8, 16, 32, dan 64 neuron. Batang-batang ini juga dilengkapi dengan *error bars* untuk menunjukkan variabilitas akurasi.

Dari diagram tersebut terlihat bahwa akurasi model relatif stabil untuk semua fungsi aktivasi, tanpa perbedaan signifikan di antara Sigmoid, Softmax, ReLU, dan Tanh. Selain itu, jumlah neuron dalam hidden layer juga tidak memberikan pengaruh besar terhadap akurasi, karena setiap jumlah neuron (dari 4 hingga 64) menghasilkan akurasi yang hampir sama dengan sedikit variasi. *Error bars* yang kecil menunjukkan konsistensi hasil model pada kombinasi fungsi aktivasi dan jumlah neuron.

Kesimpulannya, baik pemilihan fungsi aktivasi maupun jumlah neuron dalam rentang yang diuji tidak memberikan dampak signifikan terhadap akurasi model. Hal ini menunjukkan bahwa model cukup stabil terhadap perubahan kedua hyperparameter tersebut.