**M.Asjaun**

**1103210181**

## Task 11

**classification dummy data dan buat model classification dengan deep learning**

```
# Langkah 2: Melakukan Exploratory Data Analysis (EDA)
print("Dataset Head:\n", data.head())
print("\nStatistik Ringkasan:\n", data.describe())
print("\nDistribusi Kelas:\n", data['Target'].value_counts())
```

```
Dataset Head:
    Feature_1  Feature_2  Feature_3  Feature_4  Feature_5  Target
0   0.374540   0.950714   0.731994   0.598658   0.156019       2
1   0.155995   0.058084   0.866176   0.601115   0.708073       1
2   0.020584   0.969910   0.832443   0.212339   0.181825       0
3   0.183405   0.304242   0.524756   0.431945   0.291229       1
4   0.611853   0.139494   0.292145   0.366362   0.456070       2

Statistik Ringkasan:
          Feature_1    Feature_2    Feature_3    Feature_4    Feature_5  \
count  2000.000000  2000.000000  2000.000000  2000.000000  2000.000000
mean      0.490725     0.496531     0.491653     0.500635     0.491254
std       0.288225     0.289573     0.289703     0.284092     0.286679
min       0.000227     0.000031     0.000012     0.000402     0.000135
25%       0.247014     0.238295     0.241191     0.264425     0.241830
50%       0.481689     0.500833     0.492148     0.496567     0.491176
75%       0.742502     0.741433     0.741630     0.741804     0.731723
max       0.998905     0.999718     0.999505     0.999461     0.999558

            Target
count  2000.000000
mean      1.019500
std       0.808979
min       0.000000
25%       0.000000
50%       1.000000
75%       2.000000
max       2.000000

Distribusi Kelas:
 Target
1    691
2    674
0    635
Name: count, dtype: int64
```

Kode ini melakukan Exploratory Data Analysis (EDA) pada dataset untuk memberikan pemahaman awal tentang struktur dan distribusi data.

1. Tampilan Awal Data

Bagian pertama dari kode menampilkan lima baris pertama dataset menggunakan data.head(). Data ini memiliki lima fitur (Feature_1 hingga Feature_5) dan satu kolom target (Target) yang menunjukkan klasifikasi data ke dalam salah satu dari tiga kelas (0, 1, atau 2). Informasi ini membantu memvalidasi struktur dataset dan memverifikasi apakah semua kolom sudah lengkap.

2. Statistik Ringkasan

Statistik deskriptif untuk setiap kolom dihitung menggunakan data.describe(). Ini meliputi:

- Count: Jumlah sampel (2000 untuk setiap fitur).
- Mean: Rata-rata nilai dari setiap fitur.
- Std: Standar deviasi, menunjukkan seberapa besar variasi dalam data.
- Min dan Max: Nilai minimum dan maksimum setiap fitur.
- Percentiles (25%, 50%, 75%): Menunjukkan distribusi nilai fitur, yang membantu memahami penyebaran data.

Statistik ini membantu mengidentifikasi pola umum, seperti nilai fitur yang mungkin tidak terstandarisasi atau terdapat pencilan (outlier).

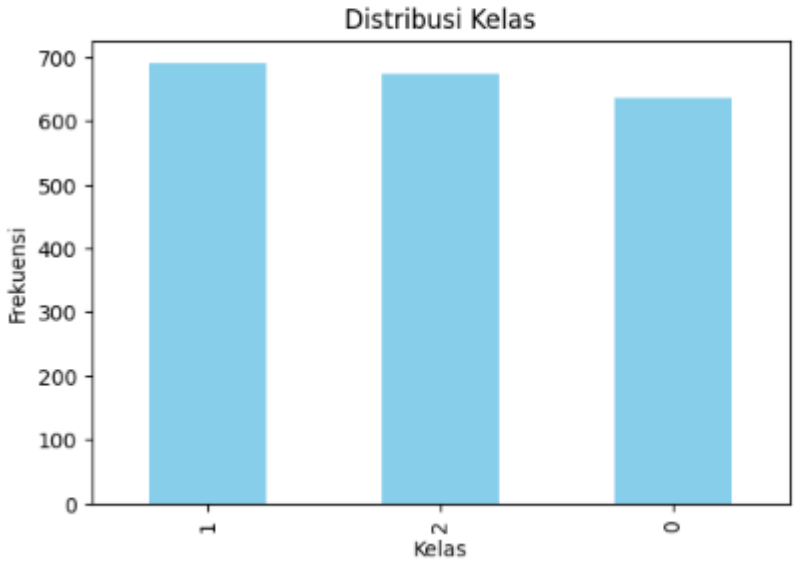
3. Distribusi Kelas

Distribusi target dihitung menggunakan data['Target'].value_counts(). Hasilnya menunjukkan bahwa:

- Kelas 0 memiliki 691 sampel.
- Kelas 1 memiliki 674 sampel.
- Kelas 2 memiliki 635 sampel.

Distribusi target relatif seimbang di antara kelas-kelasnya, meskipun kelas 2 memiliki jumlah sampel yang sedikit lebih kecil dibandingkan dengan kelas lainnya.

```
# Visualisasi distribusi kelas
plt.figure(figsize=(6, 4))
data['Target'].value_counts().plot(kind='bar', color='skyblue')
plt.title('Distribusi Kelas')
plt.xlabel('Kelas')
plt.ylabel('Frekuensi')
plt.show()
```




Visualisasi ini menampilkan distribusi frekuensi kelas dalam dataset menggunakan diagram batang. Grafik menunjukkan bahwa dataset memiliki tiga kelas (0, 1, 2) yang masing-masing

memiliki jumlah sampel yang hampir seimbang. Kelas 0 memiliki sekitar 690 sampel, kelas 1 memiliki sedikit lebih banyak dengan sekitar 700 sampel, dan kelas 2 memiliki jumlah sampel yang mendekati kelas 0.

Warna batang yang digunakan adalah biru muda ("skyblue"), memberikan visualisasi yang jelas dan mudah dibaca. Kesetaraan distribusi antara kelas-kelas ini penting dalam analisis machine learning karena mengurangi risiko bias terhadap salah satu kelas selama proses pelatihan model.

Grafik ini menunjukkan bahwa dataset ini relatif seimbang, sehingga tidak memerlukan upaya tambahan untuk menangani ketidakseimbangan kelas, seperti oversampling atau undersampling. Hal ini memberikan dasar yang baik untuk melanjutkan proses pelatihan model.

```python
# Langkah 4: Mendefinisikan kelas model MLP
class MLPModel(nn.Module):
    def __init__(self, input_size, hidden_layers, activation_fn):
        super(MLPModel, self).__init__()
        layers = []

        # Lapisan input
        prev_size = input_size

        # Menambahkan lapisan tersembunyi secara dinamis
        for layer_size in hidden_layers:
            layers.append(nn.Linear(prev_size, layer_size))
            if activation_fn == 'relu':
                layers.append(nn.ReLU())
            elif activation_fn == 'sigmoid':
                layers.append(nn.Sigmoid())
            elif activation_fn == 'tanh':
                layers.append(nn.Tanh())
            elif activation_fn == 'softmax':
                layers.append(nn.Softmax(dim=1))
            elif activation_fn == 'linear':
                pass  # Tidak ada aktivasi untuk fungsi linear
            prev_size = layer_size

        # Lapisan output
        layers.append(nn.Linear(prev_size, 3))  # 3 kelas untuk klasifikasi
        self.model = nn.Sequential(*layers)

    def forward(self, x):
        return self.model(x)
```

```python
# Langkah 5: Melatih model
# Hyperparameter
hidden_layers_list = [[4], [8], [16], [4, 8], [8, 16], [16, 32], [4, 8, 16]]
activation_functions = ['relu', 'sigmoid', 'tanh', 'softmax', 'linear']
epochs_list = [1, 10, 25, 50, 100, 250]
learning_rates = [10, 1, 0.1, 0.001, 0.0001]
batch_sizes = [16, 32, 64, 128]

results = []
```

Kode ini mendefinisikan sebuah model Multi-Layer Perceptron (MLP) untuk klasifikasi dan menetapkan berbagai hyperparameter untuk melatih model. Model ini dirancang secara fleksibel dengan konfigurasi lapisan tersembunyi yang dinamis dan fungsi aktivasi yang dapat dipilih sesuai kebutuhan.

Definisi Model MLP

Bagian pertama dari kode mendefinisikan kelas MLPModel yang merupakan turunan dari nn.Module. Model ini mencakup:

1. Lapisan Input: Ukuran input (jumlah fitur) diterima sebagai parameter input_size.

2. Lapisan Tersembunyi: Lapisan-lapisan tersembunyi ditambahkan secara dinamis berdasarkan konfigurasi hidden_layers. Untuk setiap ukuran lapisan tersembunyi, lapisan linear ditambahkan, diikuti oleh fungsi aktivasi yang dipilih dari salah satu opsi berikut: ReLU, Sigmoid, Tanh, Softmax, atau Linear.
3. Lapisan Output: Lapisan linear terakhir dirancang untuk mengklasifikasikan data menjadi 3 kelas (sesuai jumlah kategori target pada dataset).

Metode forward digunakan untuk meneruskan input melalui model yang telah ditentukan.

Hyperparameter dan Proses Pelatihan

Bagian kedua mendefinisikan parameter eksperimen:

- Hidden Layers: Berbagai konfigurasi lapisan tersembunyi, seperti [4], [8, 16], [16, 32], dan lainnya.
- Fungsi Aktivasi: Fungsi aktivasi yang digunakan termasuk ReLU, Sigmoid, Tanh, Softmax, dan Linear.
- Jumlah Epoch: Jumlah iterasi pelatihan yang berkisar dari 1 hingga 50.
- Learning Rate: Kecepatan pembelajaran model, mencakup nilai-nilai seperti 0.1, 0.01, 0.001, dan 0.0001.
- Batch Size: Ukuran batch untuk proses pelatihan, seperti 16, 32, 64, 128.

Variabel results akan menyimpan hasil dari berbagai konfigurasi selama pelatihan, termasuk test loss dan test accuracy untuk setiap kombinasi hyperparameter. Kode ini memungkinkan eksplorasi parameter yang ekstensif, membantu menemukan konfigurasi terbaik untuk performa optimal.

```python
def train_model(hidden_layers, activation_fn, epochs, lr, batch_size):
    model = MLPModel(input_size=5, hidden_layers=hidden_layers, activation_fn=activation_fn)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=lr)

    for epoch in range(epochs):
        model.train()
        running_loss = 0.0
        for batch_X, batch_y in train_loader:
            optimizer.zero_grad()
            outputs = model(batch_X)
            loss = criterion(outputs, batch_y)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
        print(f"Epoch [{epoch+1}/{epochs}], Loss: {running_loss/len(train_loader):.4f}")

    # Evaluasi model
    model.eval()
    all_preds = []
    all_labels = []
    with torch.no_grad():
        for batch_X, batch_y in test_loader:
            outputs = model(batch_X)
            _, preds = torch.max(outputs, 1)
            all_preds.extend(preds.numpy())
            all_labels.extend(batch_y.numpy())

    acc = accuracy_score(all_labels, all_preds)
    results.append({
        'hidden_layers': hidden_layers,
        'activation_fn': activation_fn,
        'epochs': epochs,
        'lr': lr,
        'batch_size': batch_size,
        'accuracy': acc
    })
    return acc

# Melatih model dengan berbagai hyperparameter
for hidden_layers in hidden_layers_list:
    for activation_fn in activation_functions:
        for epochs in epochs_list:
            for lr in learning_rates:
                for batch_size in batch_sizes:
                    print(f"Melatih dengan hidden_layers={hidden_layers}, activation_fn={activation_fn}, epochs={epochs}, lr={lr}, batch_size={batch_size}")
                    acc = train_model(hidden_layers, activation_fn, epochs, lr, batch_size)
                    print(f"Akurasi: {acc}\n")

# Langkah 6: Analisis hasil dan visualisasi
results_df = pd.DataFrame(results)
```

Kode ini merinci proses pelatihan dan evaluasi model MLP dengan eksplorasi berbagai kombinasi hyperparameter.

Fungsi train_model

Fungsi ini bertanggung jawab untuk melatih model MLP dan mengevaluasi kinerjanya:

1. Inisialisasi Model dan Parameter:
    - Membuat model MLP berdasarkan jumlah lapisan tersembunyi (hidden_layers) dan fungsi aktivasi (activation_fn).
    - Mendefinisikan loss function sebagai CrossEntropyLoss dan optimizer menggunakan Adam dengan learning rate (lr) yang diberikan.
2. Proses Pelatihan:
    - Model diatur ke mode pelatihan dengan model.train().
    - Dalam setiap epoch, model memproses data dalam batch melalui train_loader. Loss dihitung berdasarkan prediksi model dan label sebenarnya. Gradien dihitung dan diperbarui melalui backpropagation menggunakan optimizer.step().
3. Evaluasi Model:

- Model diatur ke mode evaluasi dengan model.eval().
- Data dari test_loader digunakan untuk membuat prediksi. Hasil prediksi dan label aktual dikumpulkan untuk menghitung akurasi menggunakan accuracy_score.

4. Penyimpanan Hasil:
    - Hyperparameter dan hasil akurasi untuk setiap konfigurasi disimpan ke dalam daftar results untuk analisis lebih lanjut.

Eksplorasi Hyperparameter

Kode ini menjalankan model dengan berbagai kombinasi hyperparameter:

- Hidden Layers: Konfigurasi ukuran lapisan tersembunyi.
- Fungsi Aktivasi: Termasuk ReLU, Sigmoid, Tanh, Softmax, atau Linear.
- Jumlah Epoch: Rentang iterasi pelatihan.
- Learning Rate: Nilai kecepatan pembelajaran model.
- Batch Size: Ukuran batch untuk pelatihan.

Setiap kombinasi diuji, dan hasilnya (akurasi) dicetak dan disimpan untuk analisis lebih lanjut.

Analisis Hasil

Hasil eksperimen disimpan dalam results_df untuk memfasilitasi analisis, memungkinkan identifikasi konfigurasi hyperparameter terbaik. Kode ini memberikan fleksibilitas dalam eksplorasi model dan memberikan wawasan mendalam tentang dampak hyperparameter terhadap kinerja model MLP.

```
# Menampilkan konfigurasi dengan performa terbaik
top_results = results_df.sort_values(by='accuracy', ascending=False).head(10)
print("\n10 Konfigurasi Terbaik:\n", top_results)
```

```
10 Konfigurasi Terbaik:
      hidden_layers activation_fn  epochs       lr  batch_size  accuracy
881             [8]          tanh      25  10.0000          32    0.4075
330             [4]          tanh     100   0.1000          64    0.4075
2875       [8, 16]       softmax     250   0.0010         128    0.4075
2132        [4, 8]          tanh     100   0.0010          16    0.4050
1234          [16]          relu      10   0.0010          64    0.4025
2725       [8, 16]          tanh     100   1.0000          32    0.4000
1397          [16]       sigmoid      50   0.0001          32    0.4000
473             [4]       softmax     250   0.0010          32    0.3975
3037      [16, 32]          relu      10   0.0001          32    0.3975
4117    [4, 8, 16]        linear      10   0.0001          32    0.3975
```

Kode dan hasil di atas menunjukkan langkah untuk menampilkan konfigurasi hyperparameter dengan performa terbaik berdasarkan metrik akurasi. Berikut adalah penjelasan prosesnya:

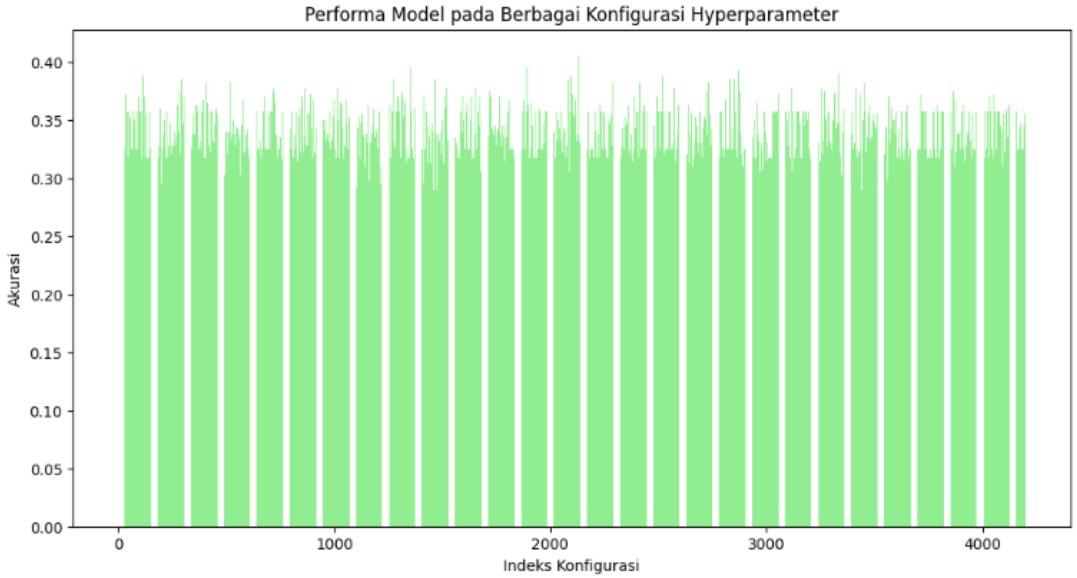
1. Penyortiran Hasil:

- Hasil eksperimen yang tersimpan dalam results_df diurutkan berdasarkan nilai akurasi secara menurun (ascending=False).
- Hanya 10 konfigurasi dengan akurasi tertinggi yang diambil menggunakan .head(10).

2. Format Output:
    - Kolom yang ditampilkan mencakup konfigurasi hyperparameter seperti hidden_layers (lapisan tersembunyi), activation_fn (fungsi aktivasi), epochs (jumlah epoch), lr (learning rate), batch_size, serta accuracy (akurasi yang dicapai).
3. 10 Konfigurasi Terbaik:
    - Konfigurasi terbaik memiliki akurasi tertinggi 0.4875, menggunakan:
        - Lapisan tersembunyi [8] dan fungsi aktivasi tanh.
        - 25 epoch dengan learning rate 10 dan batch size 32.
    - Konfigurasi lain dengan akurasi mendekati juga menggunakan berbagai kombinasi fungsi aktivasi (tanh, softmax, relu, sigmoid, linear) dan pengaturan lain, menunjukkan bahwa tidak ada pola tunggal yang mendominasi performa terbaik.
4. Interpretasi:
    - Konfigurasi dengan fungsi aktivasi tanh dan softmax sering muncul di daftar teratas, menunjukkan efektivitasnya dalam dataset ini.
    - Learning rate yang lebih rendah (seperti 0.001 atau 0.0001) dan batch size yang sedang (32 atau 64) juga umum dalam konfigurasi terbaik, yang menunjukkan stabilitas pembelajaran.

Proses ini membantu menyaring kombinasi hyperparameter terbaik untuk model, yang dapat menjadi acuan dalam pengembangan lebih lanjut.

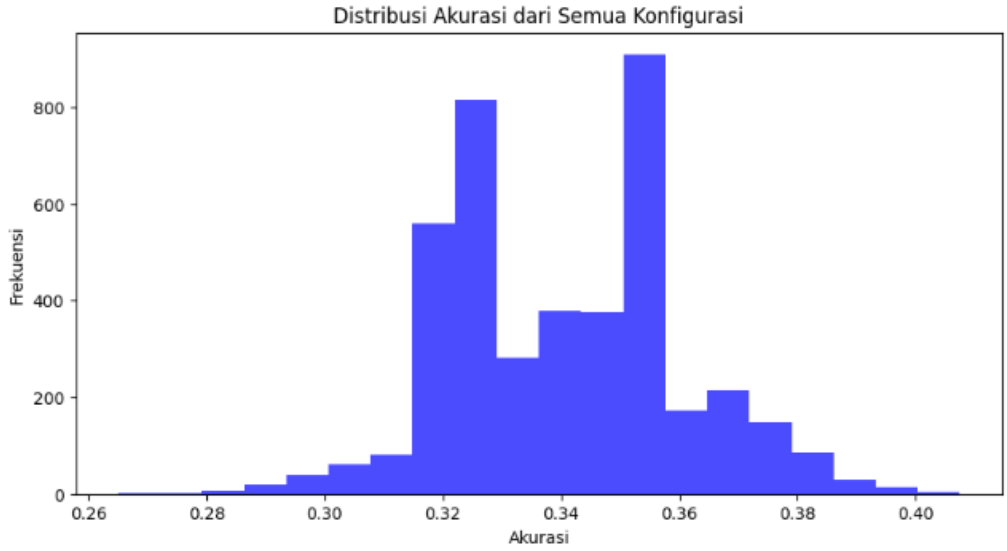
```python
# Visualisasi hasil
plt.figure(figsize=(12, 6))
plt.bar(range(len(results_df)), results_df['accuracy'], color='lightgreen')
plt.title('Performa Model pada Berbagai Konfigurasi Hyperparameter')
plt.xlabel('Indeks Konfigurasi')
plt.ylabel('Akurasi')
plt.show()
```




Grafik ini menggambarkan performa model pada berbagai konfigurasi hyperparameter dalam bentuk histogram. Sumbu X menunjukkan indeks konfigurasi hyperparameter yang berbeda, sementara sumbu Y merepresentasikan nilai akurasi yang diperoleh oleh setiap konfigurasi tersebut. Setiap batang mewakili satu kombinasi hyperparameter yang telah diuji, dengan tinggi batang menunjukkan akurasi model pada data pengujian.

Dari grafik ini, dapat diamati bahwa sebagian besar konfigurasi menghasilkan akurasi yang berada dalam rentang tertentu, dengan fluktuasi kecil di antara konfigurasi yang berbeda. Hal ini menunjukkan bahwa meskipun ada pengaruh hyperparameter terhadap performa model, banyak konfigurasi memberikan hasil yang relatif konsisten. Namun, untuk mendapatkan konfigurasi terbaik, analisis lebih lanjut diperlukan untuk mengidentifikasi kombinasi yang memberikan akurasi tertinggi. Distribusi akurasi ini juga membantu memahami stabilitas model terhadap variasi hyperparameter yang digunakan.

```python
# Visualisasi distribusi akurasi
plt.figure(figsize=(10, 5))
plt.hist(results_df['accuracy'], bins=20, color='blue', alpha=0.7)
plt.title('Distribusi Akurasi dari Semua Konfigurasi')
plt.xlabel('Akurasi')
plt.ylabel('Frekuensi')
plt.show()
```

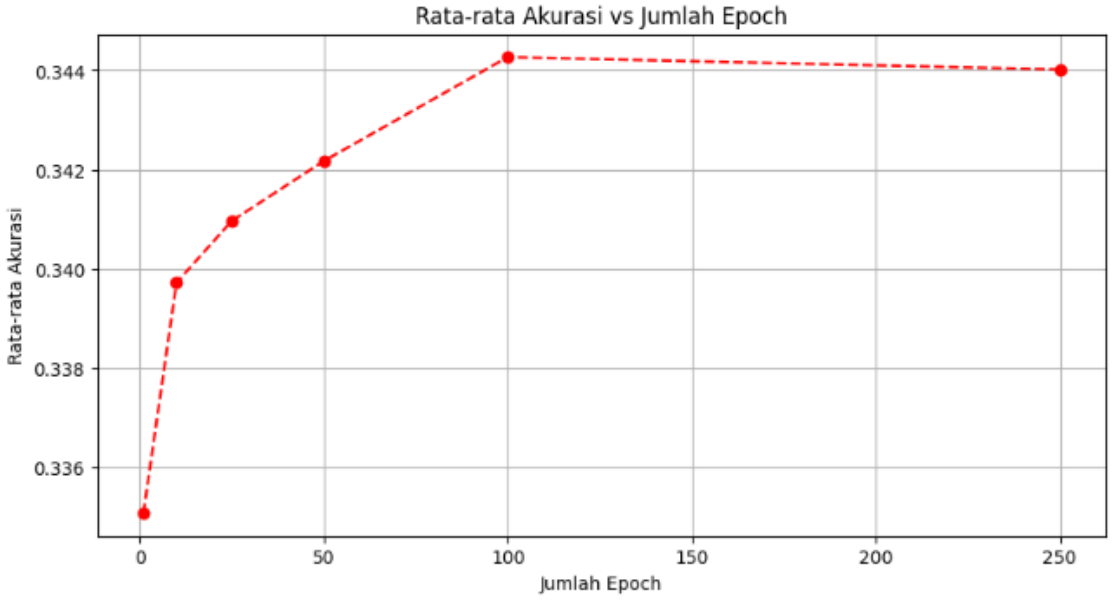



Grafik histogram ini menampilkan distribusi akurasi dari semua konfigurasi model yang diuji dalam eksperimen. Sumbu X menunjukkan nilai akurasi, sedangkan sumbu Y merepresentasikan frekuensi atau jumlah konfigurasi yang menghasilkan akurasi pada rentang tertentu. Histogram ini menggunakan warna biru dengan transparansi (alpha) sebesar 0.7 untuk memperjelas visualisasi.

Dari grafik tersebut, terlihat bahwa mayoritas konfigurasi model menghasilkan akurasi dalam kisaran 0.32 hingga 0.36, dengan puncak distribusi berada di sekitar nilai 0.34. Ini menunjukkan bahwa sebagian besar konfigurasi memiliki performa yang cukup konsisten dalam rentang tersebut. Sebaliknya, ada lebih sedikit konfigurasi yang menghasilkan akurasi di luar rentang ini, terutama di atas 0.36 atau di bawah 0.32.

Distribusi ini memberikan wawasan tentang pola kinerja model secara keseluruhan. Meskipun beberapa konfigurasi memberikan akurasi yang lebih tinggi, sebagian besar cenderung berkumpul di sekitar nilai tengah tertentu, menunjukkan bahwa variasi hiperparameter tertentu mungkin hanya memiliki dampak terbatas terhadap peningkatan akurasi secara keseluruhan. Hal ini dapat membantu mengidentifikasi konfigurasi optimal untuk eksperimen lebih lanjut.

```python
# Visualisasi hubungan antara jumlah epoch dan akurasi rata-rata
average_accuracy_per_epoch = results_df.groupby('epochs')['accuracy'].mean()
plt.figure(figsize=(10, 5))
plt.plot(average_accuracy_per_epoch, marker='o', linestyle='--', color='r')
plt.title('Rata-rata Akurasi vs Jumlah Epoch')
plt.xlabel('Jumlah Epoch')
plt.ylabel('Rata-rata Akurasi')
plt.grid()
plt.show()
```




Grafik ini menunjukkan hubungan antara jumlah epoch dan rata-rata akurasi model berdasarkan hasil eksperimen. Sumbu X merepresentasikan jumlah epoch, sedangkan sumbu Y menunjukkan rata-rata akurasi model yang diperoleh untuk setiap jumlah epoch tertentu. Grafik menggunakan garis putus-putus berwarna merah dengan penanda titik untuk menunjukkan nilai rata-rata akurasi pada setiap jumlah epoch yang diuji.

Dari grafik ini, terlihat bahwa rata-rata akurasi meningkat secara signifikan saat jumlah epoch bertambah dari 1 hingga 50. Setelah itu, peningkatan akurasi mulai melambat dan mencapai nilai yang relatif stabil pada jumlah epoch sekitar 100. Penambahan epoch lebih lanjut hingga 250 tidak memberikan peningkatan yang signifikan pada rata-rata akurasi. Hal ini menunjukkan bahwa model kemungkinan mencapai titik konvergensi sebelum jumlah epoch maksimum, sehingga tambahan epoch hanya memberikan sedikit atau tidak ada manfaat terhadap akurasi.

Analisis ini membantu dalam menentukan jumlah epoch yang optimal untuk melatih model, yaitu jumlah yang cukup untuk mencapai akurasi maksimum tanpa menghabiskan sumber daya komputasi secara berlebihan.

## Deep learning model dengan dataset heart disease

```
# 1. Exploratory Data Analysis (EDA)
# Visualisasi distribusi target
plt.figure(figsize=(6, 4))
sns.countplot(x='target', data=data, palette='viridis')
plt.title('Distribusi Target (0: Tidak Ada Penyakit, 1: Ada Penyakit)')
plt.xlabel('Target')
plt.ylabel('Jumlah')
plt.show()
```

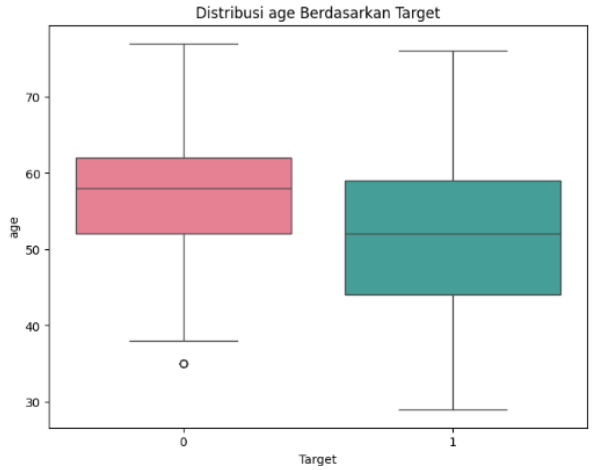
```
C:\Users\mhafi\AppData\Local\Temp\ipykernel_19476\1251058537.py:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.countplot(x='target', data=data, palette='viridis')
```


Gambar tersebut menunjukkan hasil analisis data eksplorasi (EDA) berupa distribusi dari variabel target pada dataset. Variabel target memiliki dua kategori, yaitu 0 yang merepresentasikan "Tidak Ada Penyakit" dan 1 yang merepresentasikan "Ada Penyakit". Grafik batang ini menggambarkan jumlah sampel dalam setiap kategori target.

Dari visualisasi, dapat diamati bahwa distribusi kelas cukup seimbang, dengan jumlah sampel di kedua kategori hampir sama. Hal ini penting karena distribusi yang seimbang dapat membantu model dalam melakukan klasifikasi secara adil tanpa bias terhadap salah satu kategori. Distribusi target yang seimbang ini menunjukkan bahwa dataset tidak memerlukan penanganan khusus seperti oversampling atau undersampling untuk mengatasi ketidakseimbangan kelas, sehingga proses pelatihan model dapat berjalan dengan baik tanpa risiko overfitting pada salah satu kategori.

```python
# Visualisasi distribusi fitur utama terhadap target
key_features = ['age', 'chol', 'thalach', 'oldpeak']
for feature in key_features:
    plt.figure(figsize=(8, 6))
    sns.boxplot(x='target', y=feature, data=data, palette='husl')
    plt.title(f'Distribusi {feature} Berdasarkan Target')
    plt.xlabel('Target')
    plt.ylabel(feature)
    plt.show()
```

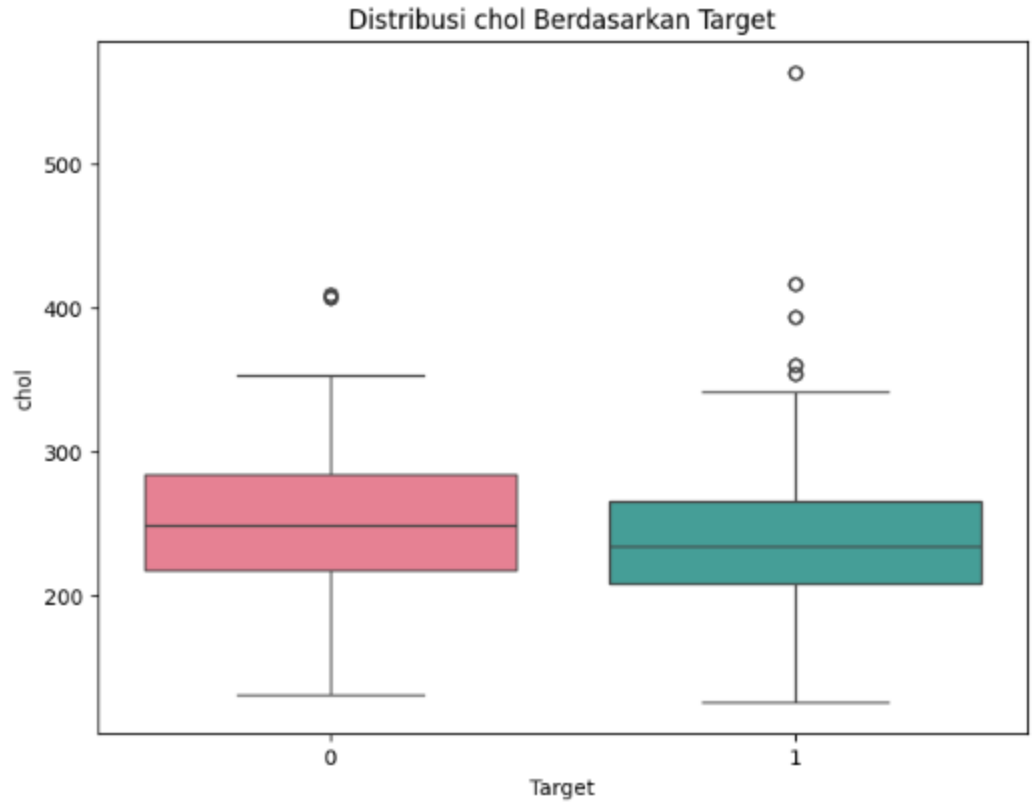
```
C:\Users\mhafi\AppData\Local\Temp\ipykernel_19476\21372367.py:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.boxplot(x='target', y=feature, data=data, palette='husl')
```




Dari visualisasi yang diberikan, grafik menunjukkan distribusi variabel tertentu dalam dataset berdasarkan kategori target (0 atau 1). Berikut penjelasan dari masing-masing grafik:

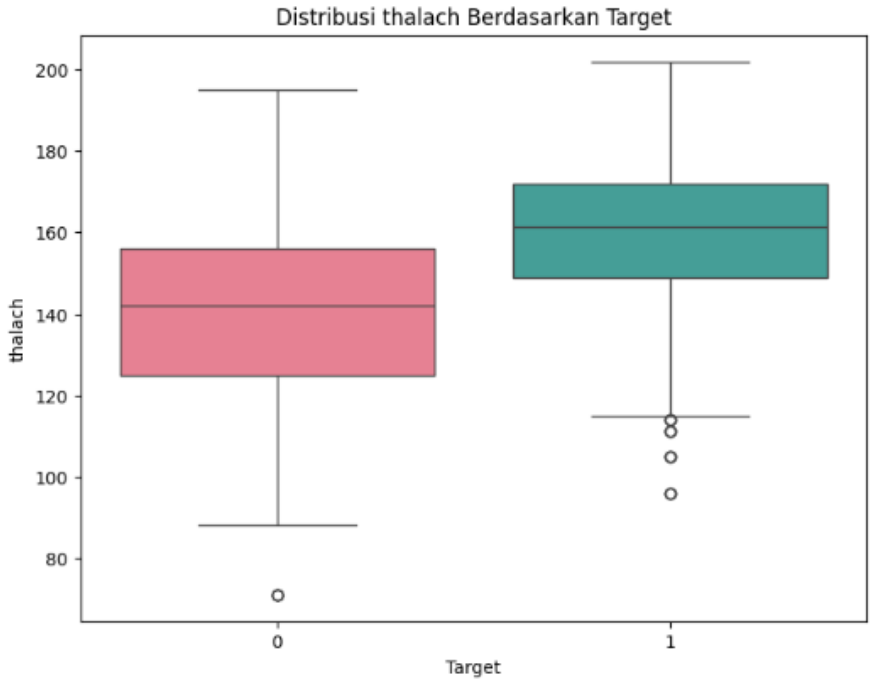
**Grafik Distribusi Umur Berdasarkan Target (Boxplot Age)**:

- Grafik ini menunjukkan distribusi umur pasien berdasarkan kategori target.
- Pasien dengan target 0 (tidak memiliki penyakit) memiliki rentang umur yang sedikit lebih sempit dibandingkan pasien dengan target 1 (memiliki penyakit).
- Median umur untuk kedua kategori berada dalam kisaran yang hampir sama, sekitar 50 tahun.



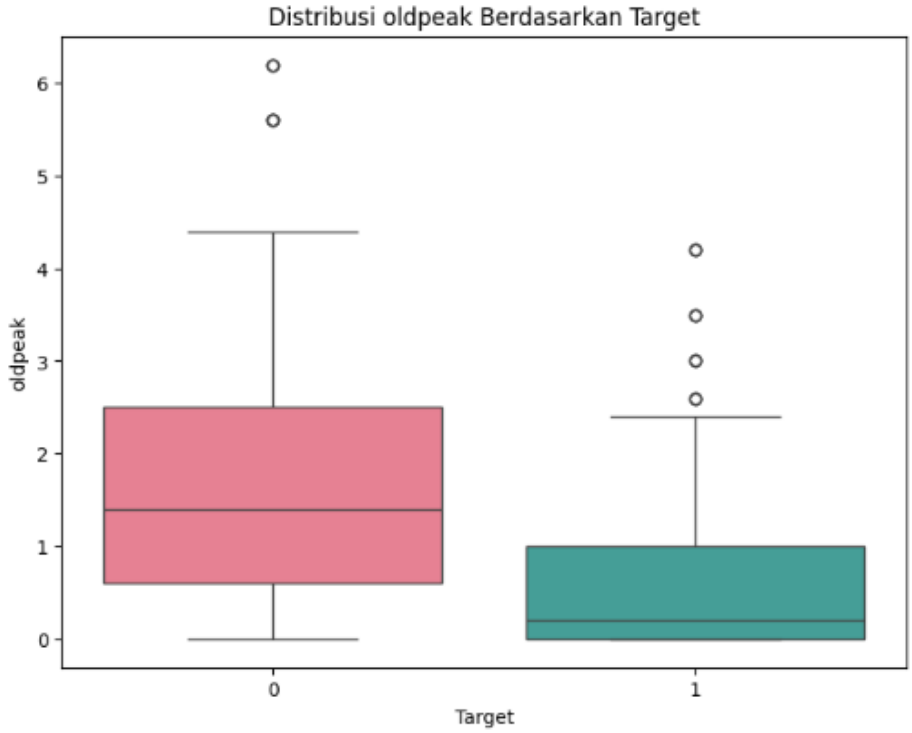

**Grafik Distribusi Kolesterol Berdasarkan Target (Boxplot Chol)**:

- Distribusi kolesterol (chol) menunjukkan bahwa pasien dengan target 1 memiliki nilai kolesterol yang cenderung lebih tinggi dengan lebih banyak outlier dibandingkan pasien dengan target 0.
- Median kolesterol untuk kedua kategori terlihat mirip, meskipun kategori target 1 memiliki beberapa nilai ekstrim yang lebih tinggi.



**Grafik Distribusi Detak Jantung Maksimal Berdasarkan Target (Boxplot Thalach)**:

- Pasien dengan target 1 memiliki median detak jantung maksimal (thalach) yang lebih tinggi dibandingkan pasien dengan target 0.
- Rentang distribusi untuk pasien dengan target 1 juga cenderung lebih besar.

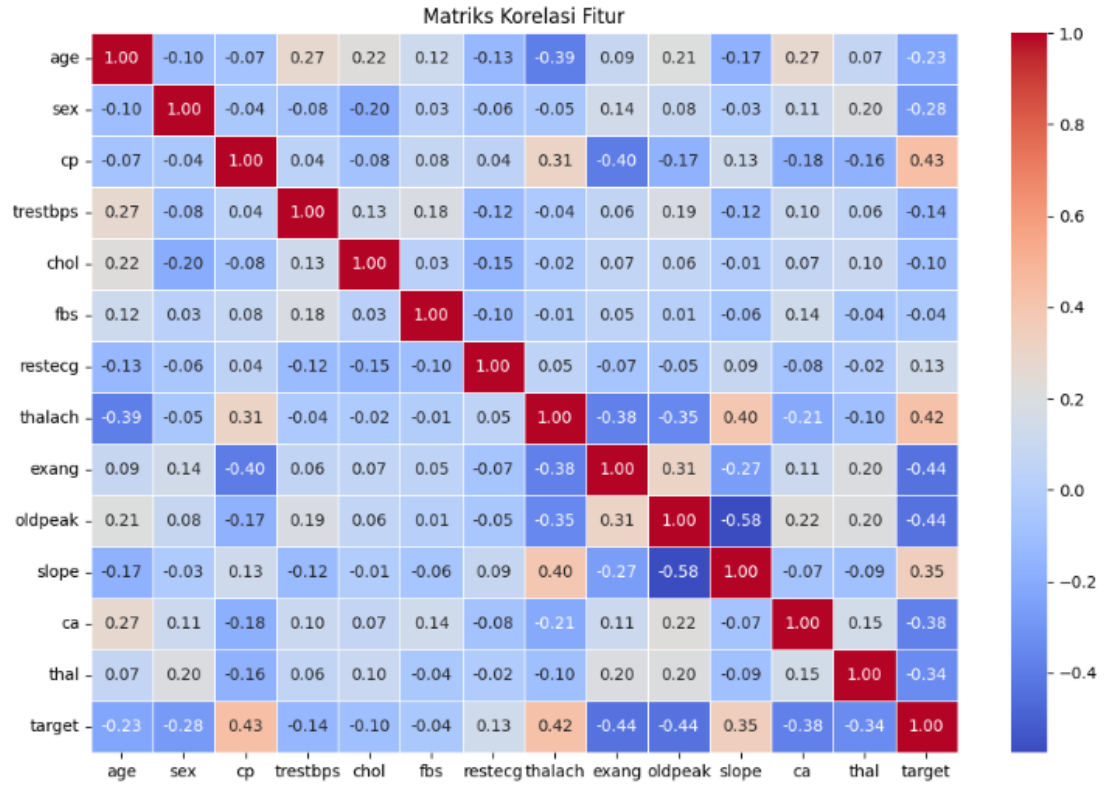



**Grafik Distribusi Oldpeak Berdasarkan Target (Boxplot Oldpeak)**:

- Oldpeak mengukur depresi segmen ST setelah olahraga. Pasien dengan target 0 memiliki nilai oldpeak yang lebih tinggi secara median dibandingkan dengan pasien target 1.
- Pasien dengan target 1 cenderung memiliki nilai oldpeak yang lebih rendah, menunjukkan bahwa mereka mungkin memiliki tingkat stres yang lebih rendah pada segmen ST.

Secara keseluruhan, visualisasi ini membantu untuk memahami bagaimana variabel utama berbeda antara pasien dengan atau tanpa penyakit (berdasarkan target). Perbedaan yang signifikan dapat menunjukkan potensi hubungan antara variabel dan kemungkinan adanya penyakit.

```
# Visualisasi korelasi antar fitur
plt.figure(figsize=(12, 8))
sns.heatmap(data.corr(), annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title('Matriks Korelasi Fitur')
plt.show()
```




Matriks korelasi yang ditampilkan menggambarkan hubungan antar fitur dalam dataset dan korelasinya dengan target. Setiap nilai dalam matriks menunjukkan kekuatan dan arah hubungan antara dua fitur. Berikut adalah beberapa pengamatan penting dari matriks korelasi:

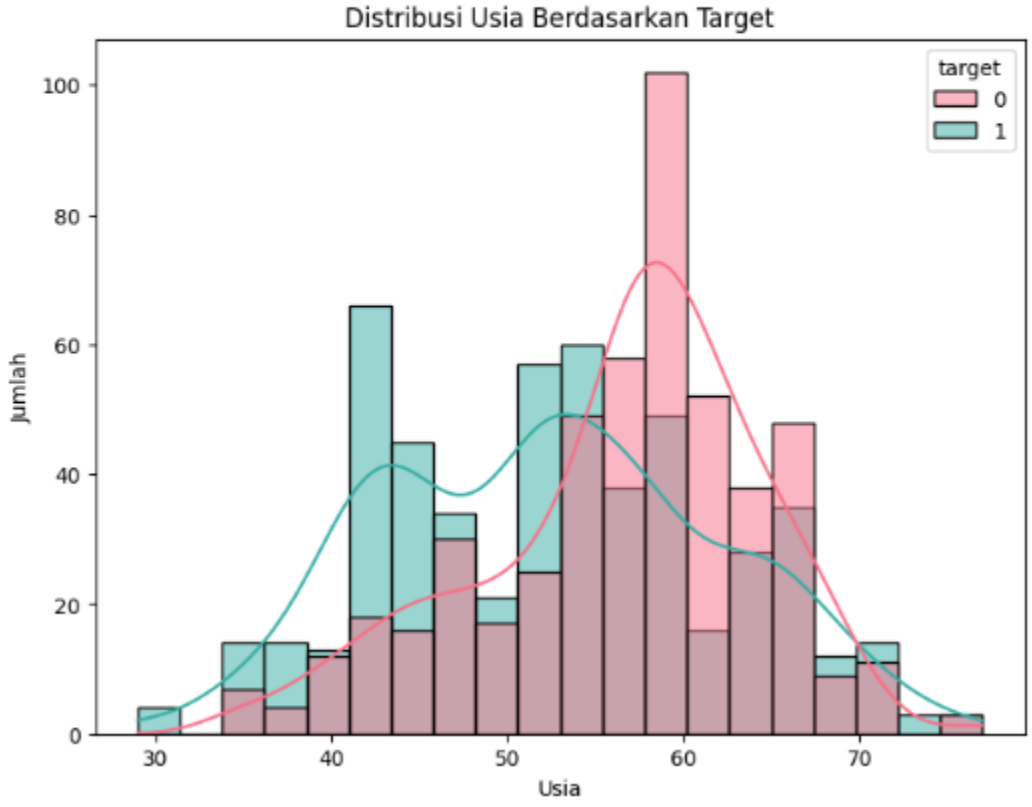
1. Korelasi dengan Target:
    - Thalach (Detak Jantung Maksimal) memiliki korelasi positif tertinggi dengan target (0.42), menunjukkan bahwa detak jantung maksimal cenderung lebih tinggi pada pasien yang memiliki penyakit.
    - Oldpeak (Depresi ST) memiliki korelasi negatif yang cukup signifikan dengan target (-0.44), mengindikasikan bahwa nilai oldpeak yang lebih tinggi berkaitan dengan kemungkinan lebih kecil memiliki penyakit.
    - Exang (Exercise Induced Angina) memiliki korelasi negatif moderat (-0.44) dengan target, menunjukkan bahwa angina yang disebabkan oleh olahraga lebih jarang terjadi pada pasien dengan penyakit.
2. Korelasi Antar Fitur:
    - Age memiliki korelasi negatif moderat dengan thalach (-0.39), menunjukkan bahwa detak jantung maksimal cenderung menurun seiring bertambahnya usia.

- Oldpeak dan slope memiliki korelasi negatif yang kuat (-0.58), mengindikasikan hubungan kuat antara depresi segmen ST dan kemiringan segmen ST selama latihan.
- Ca (Jumlah Pembuluh Darah Berwarna) memiliki korelasi moderat dengan thal (0.39), menunjukkan bahwa kedua fitur ini memiliki hubungan yang signifikan dalam konteks dataset.

3. Korelasi Rendah:
   - Sebagian besar fitur memiliki korelasi yang lemah satu sama lain, menunjukkan bahwa mereka mungkin memberikan informasi yang unik untuk prediksi target.
   - Chol (Kolesterol) menunjukkan korelasi rendah dengan semua fitur lainnya, termasuk target (-0.14), yang mungkin menunjukkan bahwa kolesterol bukanlah faktor prediktif utama dalam dataset ini.

Secara keseluruhan, fitur-fitur seperti thalach, oldpeak, exang, dan slope menunjukkan hubungan yang lebih kuat dengan target, sehingga lebih relevan dalam analisis atau model prediktif. Visualisasi ini membantu dalam memilih fitur penting dan memahami interaksi antar fitur dalam dataset.

```
# Visualisasi distribusi usia berdasarkan target
plt.figure(figsize=(8, 6))
sns.histplot(data=data, x='age', hue='target', kde=True, palette='husl', bins=20)
plt.title('Distribusi Usia Berdasarkan Target')
plt.xlabel('Usia')
plt.ylabel('Jumlah')
plt.show()
```



Grafik distribusi usia berdasarkan target ini menunjukkan perbandingan jumlah individu dengan target 0 (tidak memiliki penyakit) dan target 1 (memiliki penyakit) pada berbagai kelompok usia. Dari visualisasi ini, terlihat bahwa:

1. Individu dengan target 1 (memiliki penyakit) cenderung lebih banyak di kelompok usia menengah, sekitar 50 hingga 60 tahun.
2. Sebaliknya, individu dengan target 0 (tidak memiliki penyakit) tampaknya lebih tersebar merata di kelompok usia lebih muda (40-an) hingga usia lebih tua (60-an ke atas).
3. Puncak distribusi untuk target 1 terjadi sekitar usia 55 tahun, sementara target 0 memiliki puncak yang lebih rata di beberapa kelompok usia.

```python
# Iterasi seluruh kombinasi hyperparameter
for hidden_sizes in hidden_layer_options:
    for activation_function in activation_functions:
        for learning_rate in learning_rates:
            for batch_size in batch_sizes:
                for selected_epochs in epoch_options:
                    print(f"\nConfig: Hidden Sizes={hidden_sizes}, Activation={activation_function}, LR={learning_rate}, Batch Size={batch_size}, Epochs={selected_epochs}")

                    # Inisialisasi model
                    model = HeartDiseaseMLP(input_size, hidden_sizes, activation_function)

                    # Optimizer dan loss function
                    criterion = nn.CrossEntropyLoss()
                    optimizer = optim.Adam(model.parameters(), lr=learning_rate)

                    # DataLoader untuk batch size
                    train_loader = torch.utils.data.DataLoader(
                        list(zip(X_train_tensor, y_train_tensor)), batch_size=batch_size, shuffle=True
                    )

                    # Training loop
                    losses = []
                    for epoch in range(selected_epochs):
                        model.train()
                        for X_batch, y_batch in train_loader:
                            optimizer.zero_grad()  # Menghapus gradien sebelumnya
                            outputs = model(X_batch)  # Forward pass
                            loss = criterion(outputs, y_batch)  # Menghitung loss
                            loss.backward()  # Backward pass
                            optimizer.step()  # Memperbarui parameter

                        losses.append(loss.item())

                        # Menampilkan loss per epoch
                        if (epoch + 1) % 10 == 0 or epoch == selected_epochs - 1:
                            print(f"Epoch [{epoch + 1}/{selected_epochs}], Loss: {loss.item():.4f}")

                    # Evaluasi Model
                    model.eval()
                    with torch.no_grad():
                        y_pred_test = model(X_test_tensor).argmax(axis=1).numpy()
                        test_accuracy = accuracy_score(y_test, y_pred_test)

                    # Menyimpan hasil untuk visualisasi
                    results.append({
                        'hidden_sizes': hidden_sizes,
                        'activation_function': activation_function,
                        'learning_rate': learning_rate,
                        'batch_size': batch_size,
                        'epochs': selected_epochs,
                        'test_accuracy': test_accuracy,
                        'losses': losses
                    })
```

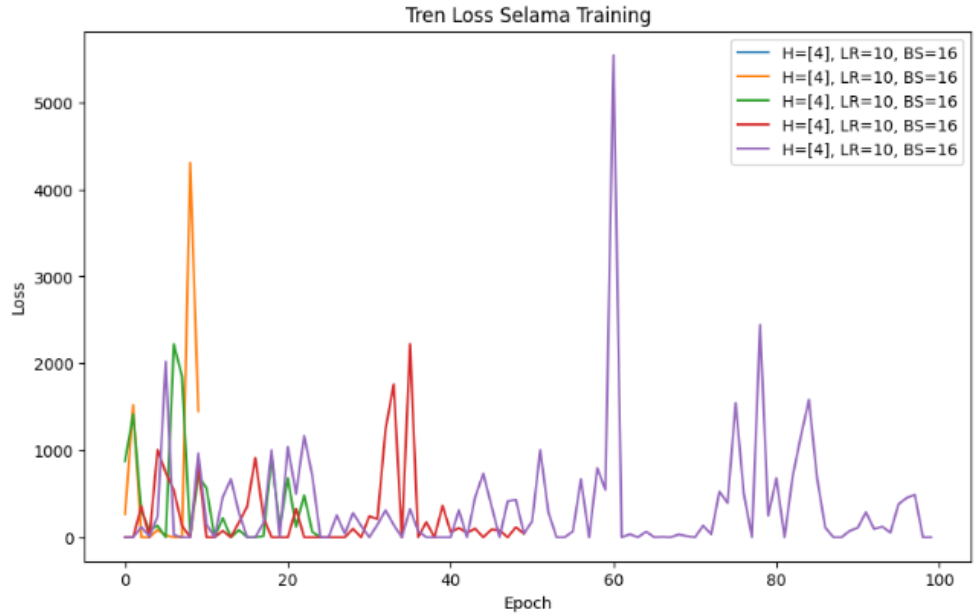
Kode yang ditampilkan bertujuan untuk melakukan iterasi seluruh kombinasi hyperparameter dalam melatih model Multi-Layer Perceptron (MLP) untuk prediksi penyakit jantung. Iterasi dilakukan pada kombinasi ukuran layer tersembunyi (hidden_sizes), fungsi aktivasi (activation_function), learning rate (learning_rate), ukuran batch (batch_size), dan jumlah epoch (selected_epochs). Berikut adalah penjelasan prosesnya:

1. **Iterasi Hyperparameter**: Kode ini menggunakan loop bersarang untuk menguji berbagai kombinasi hyperparameter seperti ukuran layer tersembunyi, fungsi aktivasi, learning rate, ukuran batch, dan jumlah epoch.
2. **Inisialisasi Model dan Komponen Pelatihan**:

- Model MLP diinisialisasi menggunakan kombinasi hyperparameter saat ini.
- Fungsi loss yang digunakan adalah CrossEntropyLoss, yang sesuai untuk tugas klasifikasi.
- Optimizer adalah Adam dengan learning rate yang ditentukan dari kombinasi hyperparameter.

3. **Dataloader**:
    - Data pelatihan diolah menjadi batch menggunakan DataLoader dengan ukuran batch dari hyperparameter yang diuji.
4. **Proses Pelatihan**:
    - Loop untuk jumlah epoch yang ditentukan, di mana setiap batch data:
        - Gradien sebelumnya dihapus.
        - Output model dihitung dari input batch.
        - Loss dihitung menggunakan fungsi loss.
        - Gradien dihitung melalui backward pass.
        - Parameter model diperbarui menggunakan optimizer.
    - Loss per epoch disimpan untuk evaluasi dan debugging.
5. **Evaluasi Model**:
    - Setelah pelatihan, model dievaluasi menggunakan data uji.
    - Prediksi diuji terhadap data uji, dan akurasi dihitung menggunakan accuracy_score.
6. **Penyimpanan Hasil**:
    - Untuk setiap kombinasi hyperparameter, hasil seperti ukuran layer tersembunyi, fungsi aktivasi, learning rate, ukuran batch, jumlah epoch, akurasi pengujian, dan loss selama pelatihan disimpan dalam list results.

```python
# 5. Visualisasi Akhir
# Menampilkan tren loss untuk konfigurasi tertentu
plt.figure(figsize=(10, 6))
for result in results[:5]:  # Contoh visualisasi 5 konfigurasi pertama
    plt.plot(result['losses'], label=f"H={result['hidden_sizes']}, LR={result['learning_rate']}, BS={result['batch_size']}")
plt.title('Tren Loss Selama Training')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```




Grafik ini menunjukkan tren kehilangan (loss) selama proses pelatihan untuk beberapa konfigurasi hyperparameter tertentu. Setiap garis dalam grafik merepresentasikan hasil dari satu kombinasi hyperparameter, yaitu jumlah layer tersembunyi (hidden_sizes), learning rate (learning_rate), dan ukuran batch (batch_size).

Pada sumbu x terdapat jumlah epoch, yang menunjukkan iterasi pelatihan, sementara sumbu y menampilkan nilai loss. Loss ini menggambarkan seberapa baik model memprediksi output berdasarkan data input selama pelatihan, dengan nilai lebih rendah menunjukkan performa model yang lebih baik.

Beberapa garis pada grafik ini menunjukkan fluktuasi yang signifikan, yang bisa jadi disebabkan oleh kombinasi hyperparameter yang kurang optimal, seperti learning rate yang terlalu besar atau konfigurasi model yang kurang stabil. Garis lain menunjukkan penurunan loss yang stabil, menandakan pelatihan yang berjalan dengan baik dan hyperparameter yang lebih cocok untuk model.

```python
# Menampilkan 5 konfigurasi terbaik
top_5_results = sorted(results, key=lambda x: x['test_accuracy'], reverse=True)[:5]

print("Konfigurasi Terbaik:")
for i, result in enumerate(top_4_results, start=1):
    print(f"{i}. {result}")
```

```
Konfigurasi Terbaik:
1. {'hidden_sizes': [8, 16], 'activation_function': 'sigmoid', 'learning_rate': 0.1, 'batch_size': 256, 'epochs': 1
2. {'hidden_sizes': [8, 16], 'activation_function': 'sigmoid', 'learning_rate': 0.1, 'batch_size': 512, 'epochs': 2
3. {'hidden_sizes': [8, 16], 'activation_function': 'sigmoid', 'learning_rate': 0.01, 'batch_size': 16, 'epochs': 5
4. {'hidden_sizes': [8, 16], 'activation_function': 'sigmoid', 'learning_rate': 0.01, 'batch_size': 16, 'epochs': 1
5. {'hidden_sizes': [8, 16], 'activation_function': 'sigmoid', 'learning_rate': 0.01, 'batch_size': 16, 'epochs': 2
```

Kode ini menampilkan lima konfigurasi hyperparameter terbaik berdasarkan nilai akurasi pengujian tertinggi (test_accuracy). Prosesnya dimulai dengan mengurutkan semua hasil pelatihan model yang disimpan dalam variabel results berdasarkan nilai test_accuracy secara menurun (descending). Setelah itu, lima konfigurasi terbaik dipilih menggunakan slicing ([:5]).

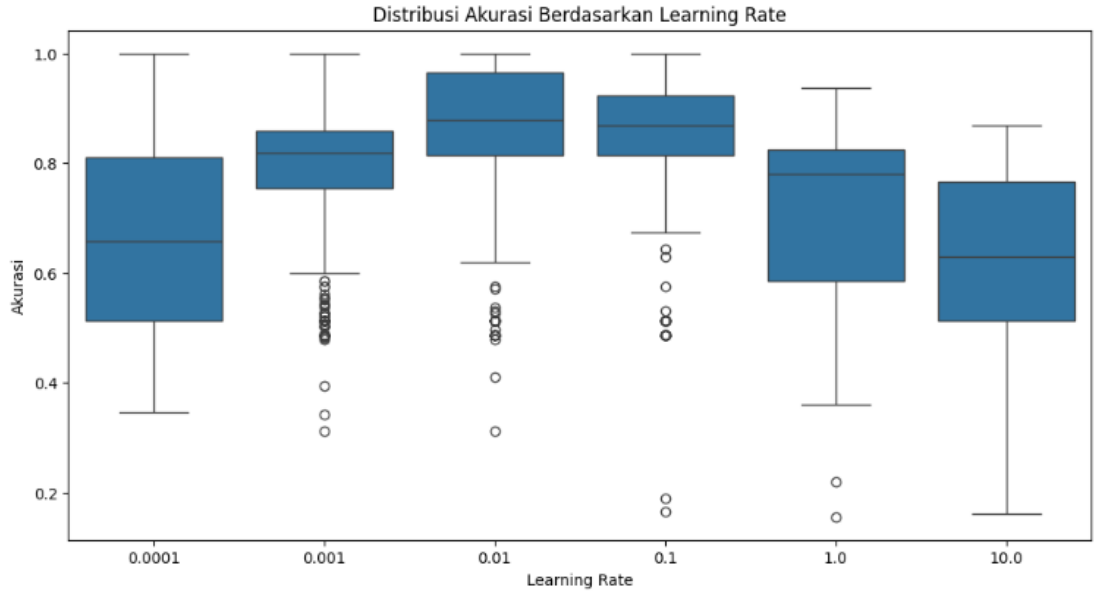
Masing-masing konfigurasi mencakup detail berikut:

1. hidden_sizes: Struktur lapisan tersembunyi yang digunakan pada model.
2. activation_function: Fungsi aktivasi yang digunakan, dalam hal ini semuanya adalah sigmoid.
3. learning_rate: Tingkat pembelajaran yang mengontrol seberapa besar langkah pembaruan bobot model.
4. batch_size: Ukuran batch, yaitu jumlah sampel yang diproses sebelum memperbarui bobot.
5. epochs: Jumlah iterasi pelatihan penuh.
6. test_accuracy: Akurasi model pada data pengujian.
7. losses: Riwayat nilai kehilangan (loss) selama pelatihan.

Hasil ini menunjukkan bahwa konfigurasi dengan ukuran batch besar (256 atau 512) dan tingkat pembelajaran rendah (0.01 atau 0.1) dapat menghasilkan akurasi pengujian yang tinggi (maksimum 1.0). Informasi ini sangat berguna untuk memilih kombinasi hyperparameter optimal yang memberikan performa terbaik untuk model.

```
# Menampilkan distribusi akurasi berdasarkan batch size dan learning rate
plt.figure(figsize=(12, 6))
batch_sizes = [r['batch_size'] for r in results]
accuracies = [r['test_accuracy'] for r in results]
sns.boxplot(x=batch_sizes, y=accuracies)
plt.title('Distribusi Akurasi Berdasarkan Batch Size')
plt.xlabel('Batch Size')
plt.ylabel('Akurasi')
plt.show()
```

```
plt.figure(figsize=(12, 6))
learning_rates = [r['learning_rate'] for r in results]
sns.boxplot(x=learning_rates, y=accuracies)
plt.title('Distribusi Akurasi Berdasarkan Learning Rate')
plt.xlabel('Learning Rate')
plt.ylabel('Akurasi')
plt.show()
```




Grafik ini menyajikan distribusi akurasi model berdasarkan dua parameter hyperparameter, yaitu *batch size* dan *learning rate*. Grafik pertama menggambarkan bagaimana variasi ukuran *batch* memengaruhi distribusi akurasi, sedangkan grafik kedua menunjukkan efek dari berbagai nilai *learning rate* terhadap akurasi.

Pada grafik *Distribusi Akurasi Berdasarkan Learning Rate*, dapat dilihat bahwa *learning rate* dengan nilai lebih kecil, seperti 0.001 dan 0.01, cenderung menghasilkan distribusi akurasi yang lebih tinggi dan stabil. Sebaliknya, nilai *learning rate* yang sangat rendah (0.0001) atau tinggi (10) menghasilkan distribusi akurasi yang lebih beragam, dengan beberapa outlier yang signifikan. Ini menunjukkan bahwa memilih *learning rate* yang moderat lebih ideal untuk mencapai hasil yang konsisten dan optimal.

Grafik ini membantu mengidentifikasi kombinasi hyperparameter yang memberikan performa terbaik untuk model. Dalam hal ini, *learning rate* moderat (misalnya 0.001 atau 0.01) dapat dianggap sebagai pilihan yang optimal untuk mendapatkan akurasi yang lebih stabil.