M.Asjaun

1103210181

Task 10

Classification model: MLP Classification

```
# Exploratory Data Analysis (EDA)
def eda(data):
    print("Data Info:")
    print(data.info())
    print("\nSummary Statistics:")
    print(data.describe())
    print("\nMissing Values:")
    print(data.isnull().sum())

    plt.figure(figsize=(10, 6))
    sns.heatmap(data.corr(), annot=True, fmt=".2f", cmap="coolwarm")
    plt.title("Correlation Heatmap")
    plt.show()

    sns.histplot(data['quality'], kde=True, bins=15)
    plt.title("Distribution of Quality")
    plt.show()

eda(data)

# Preprocess data
X = data.drop(columns=['quality'])
y = data['quality']

# Convert target to categorical (classification)
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)  # Transform target to integers (0, 1, ...)

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

Kode ini melakukan Exploratory Data Analysis (EDA) dan preprocessing data untuk sebuah masalah klasifikasi. Tahap pertama adalah analisis eksplorasi data melalui fungsi eda(data). Fungsi ini menampilkan informasi umum tentang dataset seperti struktur data (info()), statistik ringkasan (describe()), dan jumlah nilai yang hilang di setiap kolom (isnull().sum()). Selain itu, grafik heatmap korelasi dibuat untuk memvisualisasikan hubungan antar fitur dalam dataset, dan histogram dengan KDE (Kernel Density Estimation) digunakan untuk menampilkan distribusi kolom target (quality).

Setelah eksplorasi, data diproses untuk digunakan dalam model klasifikasi. Kolom target quality dipisahkan dari dataset (X untuk fitur, y untuk target). Target dikonversi menjadi nilai numerik diskret menggunakan LabelEncoder, yang sesuai dengan masalah klasifikasi, di mana setiap kelas diberi label integer (misalnya, 0, 1, 2, dst.). Selanjutnya, fitur dalam dataset (X) distandardisasi menggunakan StandardScaler agar setiap fitur memiliki skala yang seragam, yang penting untuk memastikan algoritma pembelajaran mesin bekerja secara optimal. Akhirnya, dataset dibagi menjadi data pelatihan (X_train, y_train) dan data pengujian (X_test, y_test) menggunakan train_test_split dengan 20% data dialokasikan untuk pengujian. Kode ini mempersiapkan data untuk digunakan dalam model klasifikasi sambil memastikan skala fitur seragam dan target yang sesuai untuk prediksi.

```python
# Convert to PyTorch tensors
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train, dtype=torch.long)  # For classification
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
y_test_tensor = torch.tensor(y_test, dtype=torch.long)

train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
test_dataset = TensorDataset(X_test_tensor, y_test_tensor)

# Define MLP Model class
class MLPClassifier(nn.Module):
    def __init__(self, input_size, hidden_layers, activation_fn, num_classes):
        super(MLPClassifier, self).__init__()
        layers = []
        for i, hidden_units in enumerate(hidden_layers):
            layers.append(nn.Linear(input_size if i == 0 else hidden_layers[i - 1], hidden_units))
            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 == 'linear':
                pass  # No activation
        layers.append(nn.Linear(hidden_layers[-1], num_classes))
        self.model = nn.Sequential(*layers)

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

# Hyperparameter grid
hidden_layer_configs = [[4], [8], [16], [32], [8, 16], [16, 32], [16, 32, 16]]
activations = ['linear', 'sigmoid', 'relu', 'tanh']
epochs_list = [1, 10, 25, 50, 100, 250]
learning_rates = [10, 1, 0.1, 0.01]
batch_sizes = [16, 32, 64, 128, 256]
```

Kode ini berfungsi untuk mempersiapkan data dan mendefinisikan model Multi-Layer Perceptron (MLP) menggunakan PyTorch untuk tugas klasifikasi. Data pelatihan dan pengujian yang sebelumnya telah diproses diubah menjadi PyTorch tensors menggunakan torch.tensor. Target (y_train dan y_test) dikonversi menjadi tipe long, yang diperlukan untuk klasifikasi. Dataset tensor ini kemudian dikemas dalam TensorDataset untuk digunakan dalam DataLoader selama pelatihan.

Selanjutnya, kelas MLPClassifier didefinisikan untuk membangun model MLP. Kelas ini adalah subclass dari nn.Module dan mendukung konfigurasi lapisan tersembunyi (hidden_layers), fungsi aktivasi (activation_fn), dan jumlah kelas output (num_classes). Dalam metode __init__, lapisan linear dan aktivasi ditambahkan secara dinamis sesuai dengan jumlah unit di hidden_layers. Fungsi aktivasi yang didukung termasuk ReLU, sigmoid, tanh, dan linear (tanpa aktivasi). Model akhir dirangkai menggunakan nn.Sequential, yang memudahkan pengorganisasian lapisan.

Metode forward bertanggung jawab untuk aliran data melalui model, yang mengembalikan prediksi dari lapisan terakhir.

Di bagian terakhir, terdapat konfigurasi hyperparameter untuk eksperimen, termasuk:

- hidden_layer_configs: Kombinasi jumlah unit di lapisan tersembunyi.
- activations: Fungsi aktivasi yang digunakan di antara lapisan (ReLU, sigmoid, tanh).
- epochs_list: Jumlah epoch yang diuji.
- learning_rates: Nilai learning rate untuk optimizer.
- batch_sizes: Ukuran batch yang diuji dalam proses pelatihan.

Kode ini dirancang untuk eksplorasi hyperparameter dengan fleksibilitas dalam menentukan arsitektur model, fungsi aktivasi, dan parameter pelatihan, sehingga cocok untuk optimisasi performa pada masalah klasifikasi.

```
# Training function
def train_model(model, train_loader, test_loader, epochs, lr):
    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.SGD(model.parameters(), lr=lr)

    for epoch in range(epochs):
        print(f"Epoch {epoch + 1}/{epochs}")  # Progress indicator
        model.train()
        for X_batch, y_batch in train_loader:
            optimizer.zero_grad()
            predictions = model(X_batch)
            loss = criterion(predictions, y_batch)
            loss.backward()
            optimizer.step()

    model.eval()
    with torch.no_grad():
        test_predictions = model(X_test_tensor)
        test_loss = criterion(test_predictions, y_test_tensor).item()
        test_accuracy = accuracy_score(y_test_tensor.numpy(), test_predictions.argmax(dim=1).numpy())
    return test_loss, test_accuracy
```

Kode ini mendefinisikan fungsi train_model untuk melatih dan mengevaluasi model klasifikasi menggunakan PyTorch. Fungsi ini menerima model, DataLoader untuk data pelatihan dan pengujian, jumlah epoch (epochs), serta learning rate (lr) sebagai input.

Di awal, fungsi loss CrossEntropyLoss digunakan untuk menghitung kerugian, yang sesuai untuk tugas klasifikasi multi-kelas. Optimizer yang digunakan adalah Stochastic Gradient Descent (SGD), dengan parameter model dan learning rate yang telah ditentukan.

Pada setiap epoch, model diatur ke mode pelatihan (model.train()), dan batch data dari train_loader diproses satu per satu. Untuk setiap batch:

1. Gradien diatur ulang (optimizer.zero_grad).
2. Prediksi dihasilkan menggunakan model (model(X_batch)).
3. Loss dihitung berdasarkan prediksi dan label aktual (criterion(predictions, y_batch)).
4. Backpropagation dilakukan untuk menghitung gradien (loss.backward()).
5. Parameter model diperbarui dengan optimizer (optimizer.step()).

Setelah pelatihan selesai, model diatur ke mode evaluasi (model.eval()), dan data pengujian diproses tanpa gradien (torch.no_grad()), untuk mencegah perhitungan yang tidak diperlukan. Prediksi pada data pengujian dihitung, dan nilai loss pengujian dihitung dengan criterion. Akurasi model pada data pengujian dihitung menggunakan fungsi accuracy_score, dengan membandingkan label aktual (y_test_tensor) dan prediksi yang telah dikonversi menjadi label kelas (test_predictions.argmax(dim=1)).

Akhirnya, fungsi ini mengembalikan nilai loss pengujian (test_loss) dan akurasi pengujian (test_accuracy), yang dapat digunakan untuk mengevaluasi performa model pada data yang tidak terlihat selama pelatihan. Kode ini memberikan kerangka kerja yang komprehensif untuk melatih, mengoptimalkan, dan mengevaluasi model klasifikasi.

```python
# Evaluate models
results = []

num_classes = len(np.unique(y))

for hidden_layers in hidden_layer_configs:
    for activation_fn in activations:
        for epochs in epochs_list:
            for lr in learning_rates:
                for batch_size in batch_sizes:
                    print(f"Evaluating config: hidden_layers={hidden_layers}, activation_fn={activation_fn}, epochs={epochs}, learning_rate={lr}, batch_size={batch_size}")  # Progress indicator
                    train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
                    test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

                    model = MLPClassifier(input_size=X.shape[1], hidden_layers=hidden_layers, activation_fn=activation_fn, num_classes=num_classes)
                    test_loss, test_accuracy = train_model(model, train_loader, test_loader, epochs, lr)

                    results.append({
                        'hidden_layers': hidden_layers,
                        'activation_fn': activation_fn,
                        'epochs': epochs,
                        'learning_rate': lr,
                        'batch_size': batch_size,
                        'test_loss': test_loss,
                        'test_accuracy': test_accuracy
                    })

# Save results to DataFrame
results_df = pd.DataFrame(results)
results_df.sort_values(by='test_loss', inplace=True)

print("Best Configurations:")
print(results_df.head())
```

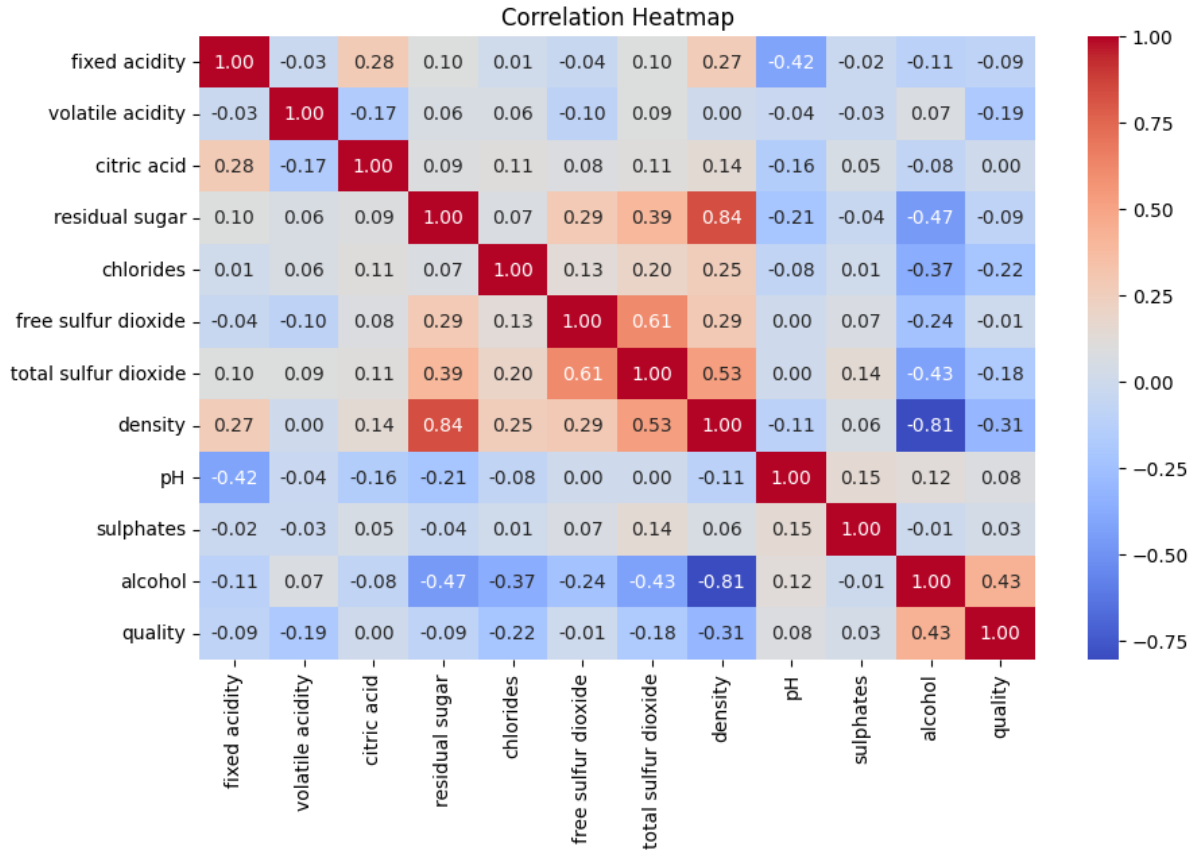
Kode ini bertujuan untuk mengevaluasi performa model Multi-Layer Perceptron (MLP) dengan berbagai kombinasi hyperparameter. Dataset yang digunakan adalah dataset klasifikasi multi-kelas, di mana jumlah kelas unik dalam target dihitung terlebih dahulu dengan np.unique(y). Evaluasi dilakukan melalui iterasi pada kombinasi konfigurasi hyperparameter yang meliputi:

1. Jumlah unit hidden layer (hidden_layers).
2. Fungsi aktivasi (activation_fn), seperti ReLU, sigmoid, atau tanh.
3. Jumlah epoch (epochs).
4. Learning rate (learning_rate).
5. Ukuran batch (batch_size).

Untuk setiap kombinasi hyperparameter, DataLoader digunakan untuk memuat data pelatihan dan pengujian dengan batch size tertentu. Model MLP dikonfigurasi menggunakan jumlah unit hidden layer, fungsi aktivasi, dan jumlah kelas dalam data. Model kemudian dilatih dan diuji menggunakan fungsi train_model, yang mengembalikan nilai test loss dan test accuracy.

Hasil setiap kombinasi hyperparameter disimpan dalam list results sebagai dictionary dengan detail konfigurasi hyperparameter dan performanya (loss dan akurasi). Setelah semua kombinasi diuji, hasil disimpan dalam bentuk DataFrame menggunakan pandas, yang mempermudah analisis lebih lanjut. DataFrame ini diurutkan berdasarkan nilai test loss untuk menampilkan konfigurasi terbaik. Konfigurasi terbaik dicetak menggunakan print, memberikan wawasan tentang kombinasi parameter yang memberikan performa terbaik pada dataset.

Dataset ini bersifat klasifikasi, di mana target adalah variabel kategori dengan lebih dari dua kelas. Tujuannya adalah memprediksi kelas target berdasarkan fitur yang ada dengan memaksimalkan akurasi dan meminimalkan loss. Kombinasi hyperparameter yang optimal membantu meningkatkan performa prediksi model.

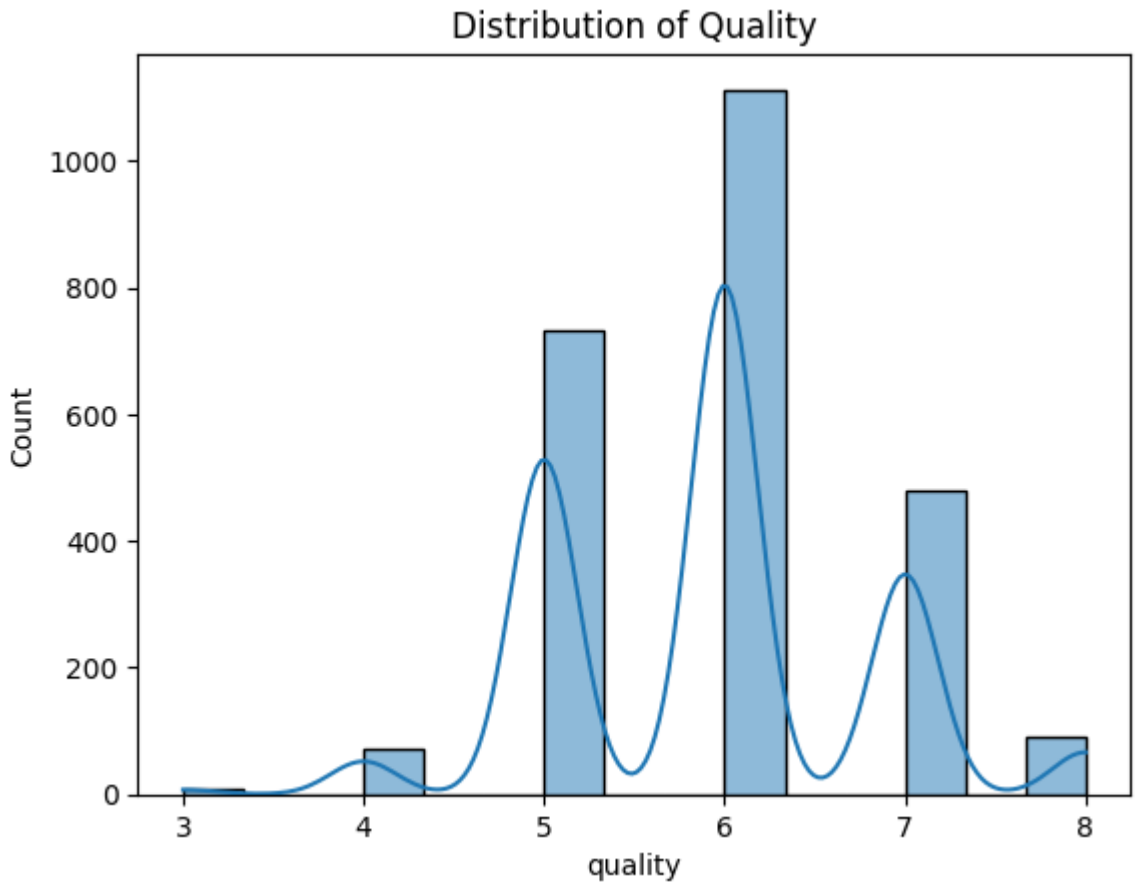


Heatmap korelasi ini menunjukkan hubungan linier antara berbagai fitur dalam dataset dan target variabel quality. Korelasi diukur dengan nilai antara -1 hingga 1, di mana nilai positif tinggi menunjukkan hubungan positif yang kuat, nilai negatif tinggi menunjukkan hubungan negatif kuat, dan nilai mendekati nol menunjukkan hubungan yang lemah.

Dari heatmap ini, beberapa hubungan signifikan terlihat. Misalnya, fitur alcohol memiliki korelasi positif tertinggi dengan quality (0.43), menunjukkan bahwa kandungan alkohol yang lebih tinggi cenderung dikaitkan dengan kualitas yang lebih baik. Sebaliknya, volatile acidity menunjukkan korelasi negatif yang relatif moderat (-0.19), menandakan bahwa tingkat keasaman volatil yang lebih tinggi dapat berdampak buruk pada kualitas.

Hubungan antar fitur lain, seperti antara density dan residual sugar (0.84), menunjukkan korelasi positif kuat, yang berarti fitur-fitur tersebut saling terkait erat. Namun, banyak fitur lain, seperti chlorides dan quality, menunjukkan korelasi yang sangat rendah (-0.22), yang berarti fitur tersebut memiliki pengaruh yang lemah terhadap target.

Secara keseluruhan, heatmap ini membantu mengidentifikasi fitur-fitur penting yang memiliki hubungan signifikan dengan kualitas, yang dapat membantu dalam pemilihan fitur untuk model prediktif.



Grafik ini menunjukkan distribusi variabel quality dalam dataset, yang merepresentasikan tingkat kualitas suatu produk atau objek (misalnya, anggur). Sumbu horizontal (x) menunjukkan nilai kualitas, sementara sumbu vertikal (y) menunjukkan jumlah observasi untuk setiap tingkat kualitas. Grafik ini dilengkapi dengan kurva KDE (Kernel Density Estimation) untuk menunjukkan pola distribusi secara kontinu.

Dari grafik, terlihat bahwa nilai quality berkisar antara 3 hingga 8, dengan mayoritas observasi terkonsentrasi pada nilai 5, 6, dan 7. Nilai kualitas 6 memiliki jumlah observasi tertinggi, menunjukkan bahwa produk dengan kualitas ini paling umum dalam dataset. Sementara itu, nilai 3 dan 8 memiliki jumlah observasi yang sangat sedikit, menandakan bahwa tingkat kualitas ini jarang terjadi.

Distribusi ini cenderung tidak simetris, dengan konsentrasi yang lebih besar pada nilai kualitas menengah. Pola ini penting untuk memahami bias distribusi target, yang mungkin memengaruhi kinerja model klasifikasi jika tidak ditangani dengan baik. Misalnya, kelas dengan jumlah data yang lebih kecil (seperti 3 dan 8) mungkin lebih sulit diprediksi oleh model karena ketidakseimbangan data.

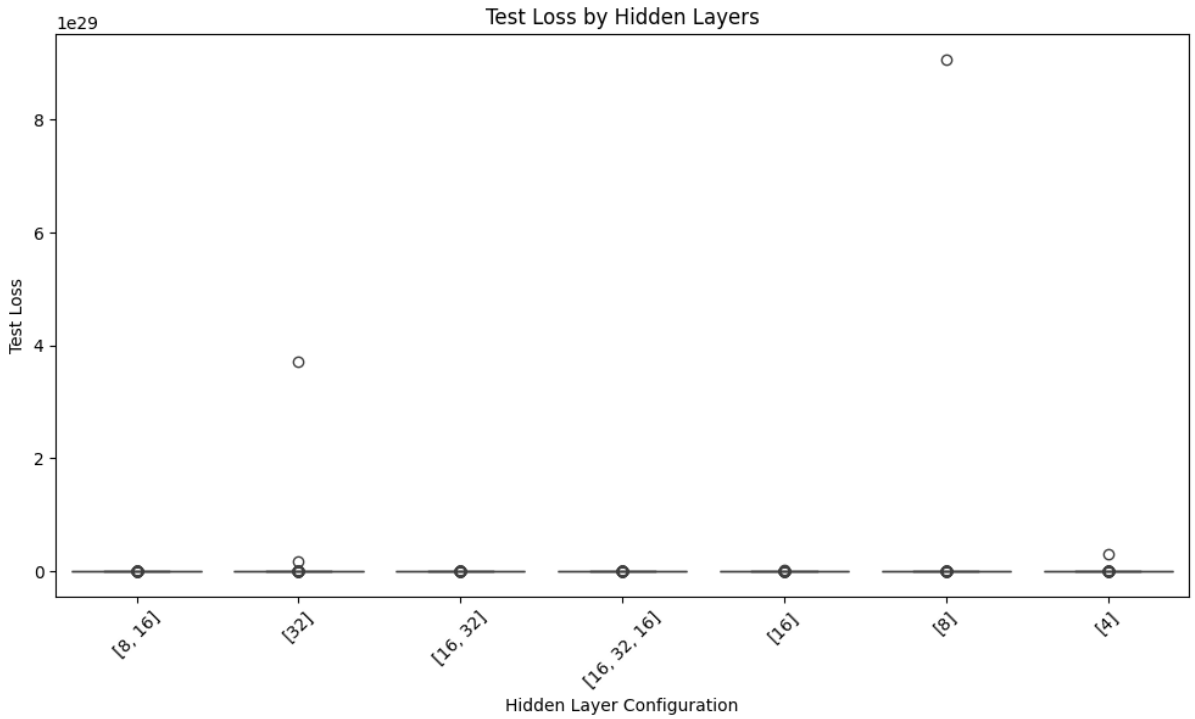
```
Best Configurations:
     hidden_layers activation_fn  epochs  learning_rate  batch_size  \
2210        [8, 16]          relu      25           0.10          16
1913           [32]          tanh     250           0.10         128
2853       [16, 32]          tanh     100           0.10         128
3356  [16, 32, 16]          tanh     250           0.01          32
1436           [16]          tanh     250           0.01          32

      test_loss  test_accuracy
2210   1.035135          0.550
1913   1.042184          0.542
2853   1.042929          0.556
3356   1.043724          0.546
1436   1.047385          0.548
```

Tabel ini menunjukkan lima konfigurasi terbaik untuk model berdasarkan kombinasi hyperparameter, yang diurutkan berdasarkan nilai **test loss** dan **test accuracy**. Hyperparameter yang dieksplorasi meliputi ukuran hidden layers, fungsi aktivasi, jumlah epoch, learning rate, dan batch size. Kolom terakhir menunjukkan performa model berupa nilai test loss dan test accuracy.

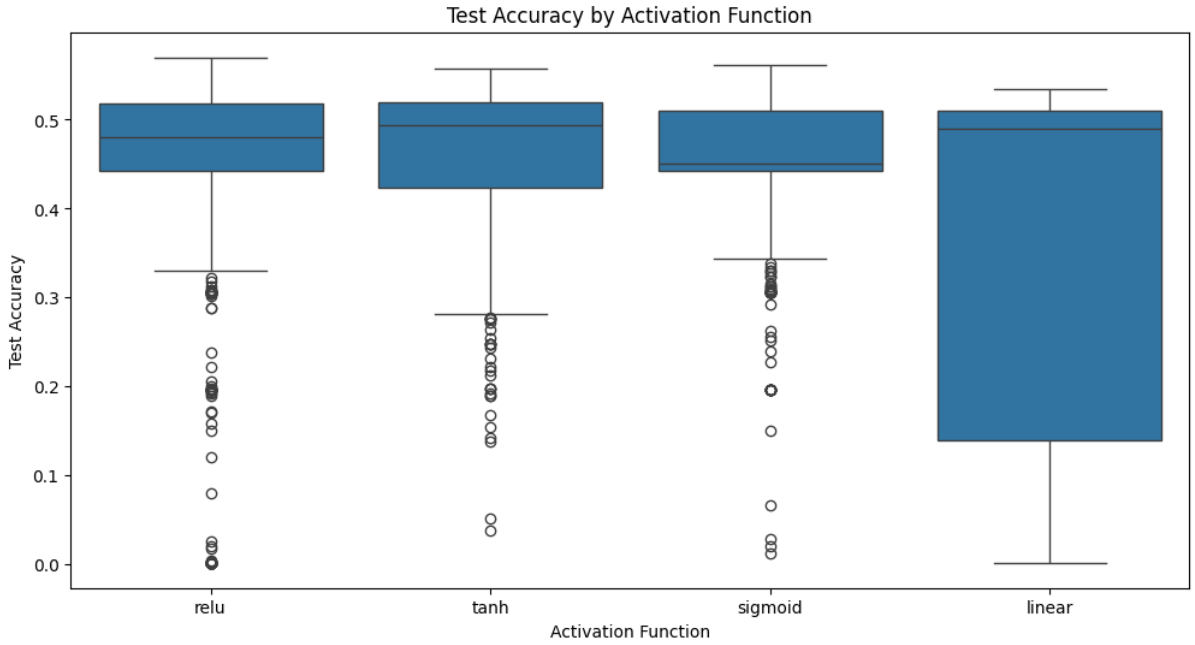
1. **Konfigurasi Terbaik** memiliki hidden layers [8, 16], fungsi aktivasi relu, 25 epoch, learning rate 0.10, dan batch size 16. Konfigurasi ini menghasilkan test loss 1.035 dan test accuracy 0.550, menunjukkan performa terbaik di antara semua kombinasi.
2. Konfigurasi kedua menggunakan hidden layer [32], fungsi aktivasi tanh, 250 epoch, learning rate 0.10, dan batch size 128, dengan test loss 1.042 dan test accuracy 0.542.
3. Konfigurasi ketiga menggunakan hidden layers [16, 32], fungsi aktivasi tanh, 100 epoch, learning rate 0.10, dan batch size 128, menghasilkan test loss 1.042 dan test accuracy 0.556, yang merupakan akurasi tertinggi di tabel ini.
4. Konfigurasi keempat memiliki hidden layers [16, 32, 16], fungsi aktivasi tanh, 250 epoch, learning rate 0.01, dan batch size 32, dengan test loss 1.043 dan test accuracy 0.546.
5. Konfigurasi kelima menggunakan hidden layers [16], fungsi aktivasi tanh, 250 epoch, learning rate 0.01, dan batch size 32, menghasilkan test loss 1.047 dan test accuracy 0.548.

Secara umum, konfigurasi dengan fungsi aktivasi tanh mendominasi hasil terbaik, sedangkan konfigurasi dengan relu memberikan test loss terendah. Learning rate yang lebih besar (0.10) dengan batch size lebih kecil (16 atau 32) cenderung menghasilkan performa yang lebih baik. Hasil ini memberikan wawasan tentang kombinasi optimal hyperparameter untuk model ini.



1. Test Loss by Hidden Layers

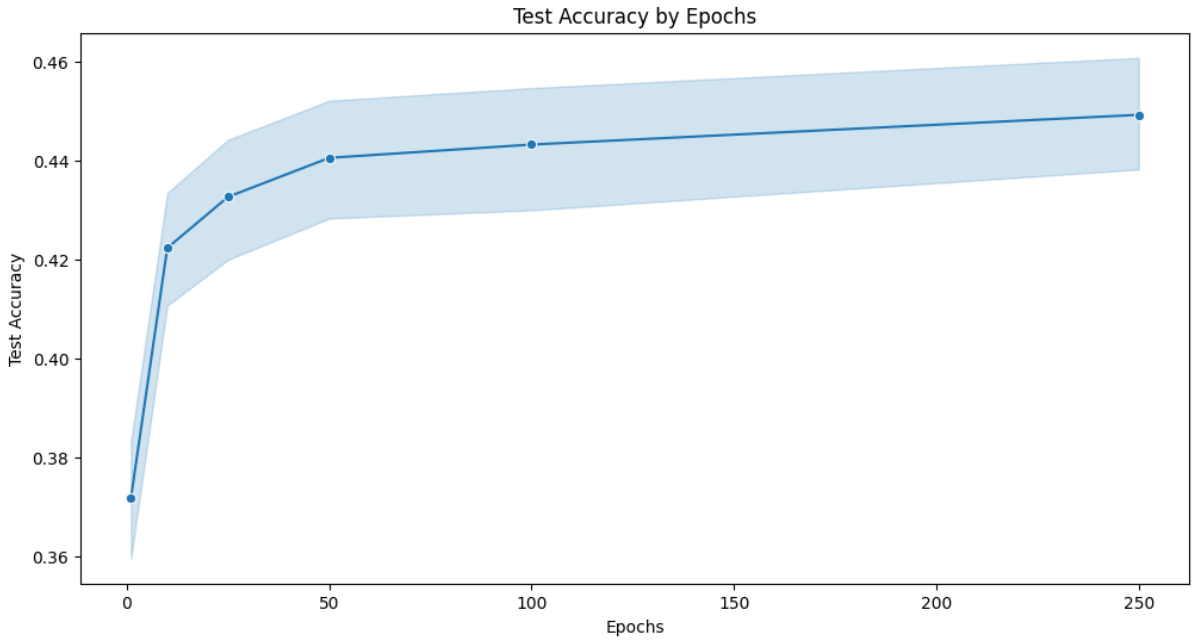
Grafik ini menunjukkan variasi nilai **test loss** untuk setiap konfigurasi hidden layer. Sebagian besar konfigurasi menghasilkan nilai test loss yang rendah, kecuali beberapa outlier, seperti hidden layer [32], yang memiliki nilai test loss sangat tinggi. Hal ini menunjukkan bahwa pemilihan ukuran hidden layer dapat secara signifikan memengaruhi performa model, dan beberapa konfigurasi mungkin tidak optimal.



Test Accuracy by Activation Function

Grafik ini membandingkan akurasi pengujian (**test accuracy**) untuk berbagai fungsi aktivasi (ReLU, tanh, sigmoid, linear). ReLU dan tanh cenderung menghasilkan akurasi yang lebih konsisten, dengan distribusi yang lebih terpusat pada nilai menengah ke atas. Sigmoid menunjukkan variasi yang lebih besar, dengan beberapa konfigurasi memberikan akurasi sangat rendah. Linear memiliki variasi

terluas, menunjukkan kinerja yang tidak konsisten, dengan beberapa konfigurasi berkinerja baik dan lainnya sangat buruk.

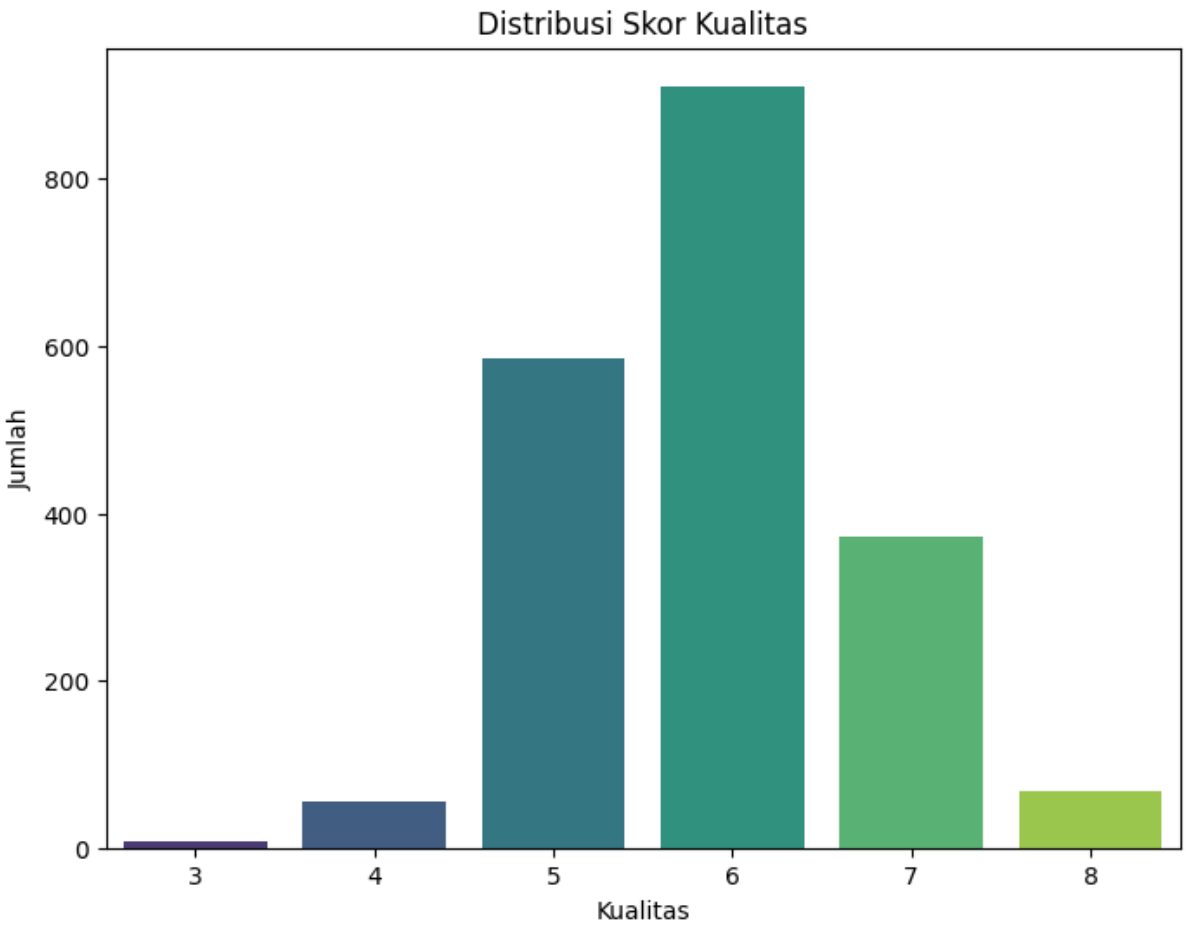


Test Accuracy by Epochs
Grafik ini menunjukkan bagaimana akurasi pengujian berubah seiring dengan peningkatan jumlah epoch. Akurasi meningkat secara signifikan pada awal pelatihan (0–50 epoch) dan mulai stabil setelah 100 epoch. Hal ini menunjukkan bahwa sebagian besar peningkatan performa terjadi di awal pelatihan, sementara peningkatan lebih lanjut menjadi semakin kecil dengan bertambahnya jumlah epoch. Area bayangan di sekitar garis menunjukkan variabilitas akurasi di berbagai konfigurasi, yang cenderung menurun seiring stabilisasi model.

Secara keseluruhan, grafik-grafik ini menunjukkan bahwa pemilihan hidden layer, fungsi aktivasi, dan jumlah epoch memiliki dampak besar pada performa model, dengan beberapa kombinasi memberikan hasil yang jauh lebih baik daripada yang lain. Pemilihan konfigurasi yang tepat sangat penting untuk mengoptimalkan hasil model.

Regression Model: MLP Regression

```
# Inspect dataset
# Menampilkan distribusi target 'quality'
plt.figure(figsize=(8, 6))
sns.countplot(x='quality', data=data, palette='viridis')
plt.title("Distribusi Skor Kualitas")
plt.xlabel("Kualitas")
plt.ylabel("Jumlah")
plt.show()
```

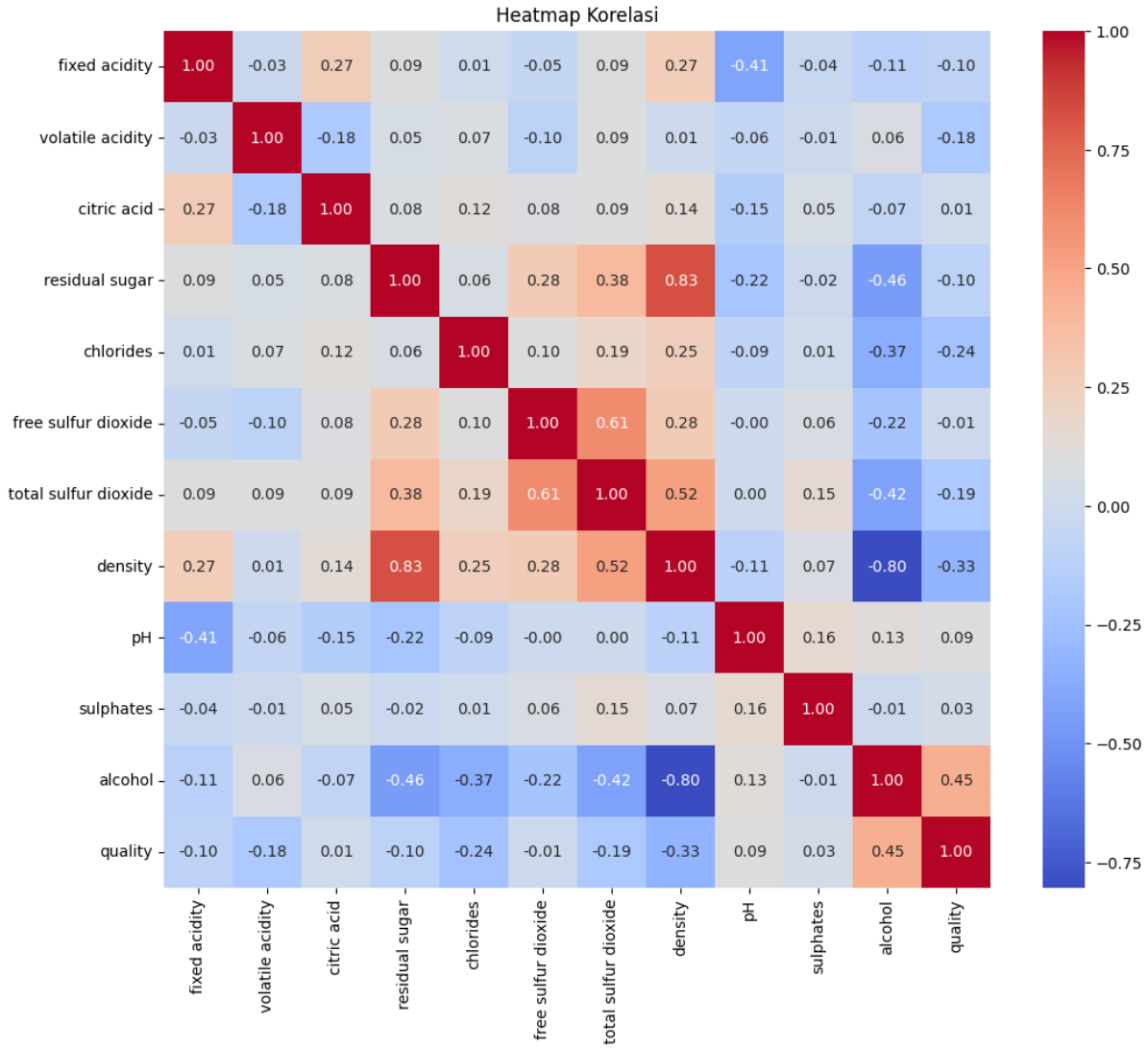


Grafik ini menunjukkan distribusi nilai variabel target quality dalam dataset, yang menggambarkan tingkat kualitas produk (seperti anggur) berdasarkan skor tertentu. Sumbu horizontal (x) menunjukkan nilai kualitas dari 3 hingga 8, sementara sumbu vertikal (y) menunjukkan jumlah observasi untuk setiap skor kualitas.

Dari grafik, terlihat bahwa nilai kualitas 6 memiliki frekuensi tertinggi dengan lebih dari 900 observasi, diikuti oleh kualitas 5, yang memiliki sekitar 600 observasi. Kualitas 7 juga cukup umum, sementara kualitas ekstrem seperti 3, 4, dan 8 memiliki jumlah observasi yang jauh lebih sedikit. Hal ini menunjukkan bahwa data cenderung terkonsentrasi pada skor kualitas menengah (5, 6, dan 7), sementara skor kualitas sangat rendah atau sangat tinggi jarang ditemukan.

Distribusi ini mengindikasikan adanya ketidakseimbangan data, di mana kelas dengan skor kualitas ekstrem memiliki jumlah data yang jauh lebih kecil dibandingkan kelas menengah. Ketidakseimbangan ini penting untuk diperhatikan dalam pemodelan, terutama pada klasifikasi, karena model dapat cenderung memberikan prediksi bias terhadap kelas mayoritas. Strategi seperti oversampling, undersampling, atau penggunaan metode penyeimbangan data lainnya mungkin diperlukan untuk mengatasi masalah ini.

```
# Correlation heatmap
# Membuat heatmap korelasi untuk fitur
plt.figure(figsize=(12, 10))
correlation = data.corr()
sns.heatmap(correlation, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Heatmap Korelasi")
plt.show()
```



Kode ini membuat **heatmap korelasi** untuk menampilkan hubungan antar fitur dalam dataset. Heatmap dihasilkan menggunakan fungsi data.corr() untuk menghitung koefisien korelasi Pearson antara semua fitur numerik. Grafik menggunakan palet warna coolwarm, dengan nilai korelasi ditampilkan dalam anotasi setiap sel, memberikan gambaran visual tentang hubungan linier antara fitur.

Dari heatmap, kita dapat mengidentifikasi hubungan utama antara fitur dan target quality. Fitur alcohol memiliki korelasi positif tertinggi dengan quality (0.45), menunjukkan bahwa kandungan alkohol yang lebih tinggi cenderung dikaitkan dengan kualitas yang lebih baik. Sebaliknya, fitur seperti density (-0.33) dan volatile acidity (-0.18) memiliki korelasi negatif, mengindikasikan bahwa peningkatan pada fitur ini cenderung berdampak buruk pada kualitas.

Selain itu, beberapa fitur menunjukkan hubungan saling terkait yang kuat, seperti antara density dan residual sugar (0.83) atau antara total sulfur dioxide dan free sulfur dioxide (0.61). Korelasi positif yang kuat ini menunjukkan bahwa fitur-fitur tersebut memiliki hubungan yang erat, yang dapat menyebabkan redundansi dalam model prediktif.

Secara keseluruhan, heatmap ini memberikan wawasan tentang hubungan penting yang dapat memengaruhi performa model, membantu dalam pemilihan fitur, dan mengidentifikasi kemungkinan fitur redundan yang dapat disederhanakan untuk meningkatkan efisiensi model.

```
# Define MLP model
# Mendefinisikan model MLP
class MLPClassifier(nn.Module):
    def __init__(self, input_size, hidden_sizes, output_size, activation):
        super(MLPClassifier, self).__init__()
        layers = []
        current_size = input_size
        for hidden_size in hidden_sizes:
            layers.append(nn.Linear(current_size, hidden_size))
            layers.append(activation)
            current_size = hidden_size
        layers.append(nn.Linear(current_size, output_size))
        self.model = nn.Sequential(*layers)

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

```
# Define experiment parameters
# Parameter eksperimen
hidden_layer_configurations = [[4], [8, 16], [32, 64, 64]]
activations = [nn.ReLU(), nn.Sigmoid(), nn.Tanh(), nn.Softmax(dim=1)]
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]
```

Definisi Model MLP

Kelas MLPClassifier adalah subclass dari nn.Module yang digunakan untuk membuat arsitektur MLP. Metode __init__ menerima beberapa parameter:

- **input_size**: Jumlah fitur input.
- **hidden_sizes**: Daftar yang mendefinisikan ukuran setiap lapisan tersembunyi.
- **output_size**: Jumlah kelas target.
- **activation**: Fungsi aktivasi yang digunakan di antara lapisan.

Model ini dibuat dengan menambahkan lapisan linear (fully connected) sesuai dengan ukuran hidden layer yang ditentukan. Fungsi aktivasi diterapkan di antara lapisan linear untuk menambah non-linearitas. Akhirnya, lapisan output linear ditambahkan untuk memetakan ke jumlah kelas target. Semua lapisan dirangkai dalam nn.Sequential untuk mempermudah eksekusi aliran data. Metode forward mendefinisikan bagaimana data melewati model untuk menghasilkan prediksi.

Parameter Eksperimen

Beberapa hyperparameter ditentukan untuk eksperimen:

1. **hidden_layer_configurations**: Berbagai kombinasi ukuran lapisan tersembunyi (contoh: [4], [8, 16], [32, 64, 64]).
2. **activations**: Beragam fungsi aktivasi seperti ReLU, sigmoid, tanh, dan softmax.
3. **epochs_list**: Daftar jumlah epoch yang akan diuji (misalnya, 1, 10, 25, hingga 250).
4. **learning_rates**: Berbagai nilai learning rate (misalnya, 10, 1, 0.1, hingga 0.0001).
5. **batch_sizes**: Ukuran batch yang bervariasi dari 16 hingga 512.

```
# Experiment results storage
# Penyimpanan hasil eksperimen
results = []

for hidden_sizes in hidden_layer_configurations:
    for activation in activations:
        for lr in learning_rates:
            for batch_size in batch_sizes:
                # Prepare DataLoader
                train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
                test_dataset = TensorDataset(X_test_tensor, y_test_tensor)
                train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
                test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

                # Initialize model
                input_size = X_train.shape[1]
                output_size = len(label_encoder.classes_)
                model = MLPClassifier(input_size, hidden_sizes, output_size, activation)

                # Define loss and optimizer
                criterion = nn.CrossEntropyLoss()
                optimizer = optim.Adam(model.parameters(), lr=lr)
```

```
# Train model
for epochs in epochs_list:
    model.train()
    start_time = time.time()
    for epoch in range(epochs):
        running_loss = 0.0
        for inputs, labels in train_loader:
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
        elapsed_time = time.time() - start_time
        print(f"Hidden Sizes: {hidden_sizes}, Activation: {activation.__class__.__name__}, Learning Rate: {lr}, Batch Size: {batch_size}, Epoch: {epoch + 1}/{epochs}, Loss: {running_loss / len(train_loader):.4f}, Time: {elapsed_time:.2f}s")

    # Evaluate model
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for inputs, labels in test_loader:
            outputs = model(inputs)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    test_accuracy = correct / total * 100

    # Save results
    results.append({
        'hidden_sizes': hidden_sizes,
        'activation': activation.__class__.__name__,
        'learning_rate': lr,
        'batch_size': batch_size,
        'epochs': epochs,
        'test_accuracy': test_accuracy
    })
```

1. Penyimpanan Hasil Eksperimen

Variabel results adalah sebuah list yang akan digunakan untuk menyimpan hasil setiap kombinasi hyperparameter, termasuk ukuran hidden layer (hidden_sizes), fungsi aktivasi (activation), learning rate (lr), ukuran batch (batch_size), jumlah epoch, dan akurasi pengujian.

2. Iterasi Hyperparameter

Loop diatur untuk mengevaluasi kombinasi:

- **Ukuran hidden layer** dari hidden_layer_configurations.
- **Fungsi aktivasi** seperti ReLU, sigmoid, atau tanh.
- **Learning rate** dari berbagai nilai.
- **Ukuran batch** untuk DataLoader.

3. Persiapan DataLoader

Dataset pelatihan dan pengujian dibungkus menggunakan TensorDataset, lalu dimuat ke dalam DataLoader dengan ukuran batch yang ditentukan. train_loader digunakan untuk pelatihan dengan shuffle=True, sementara test_loader digunakan untuk evaluasi tanpa pengacakan.

4. Inisialisasi Model, Loss, dan Optimizer

Model MLP dibuat berdasarkan kombinasi parameter yang sedang dievaluasi. Fungsi loss yang digunakan adalah CrossEntropyLoss, cocok untuk tugas klasifikasi multi-kelas. Optimizer yang digunakan adalah Adam, dengan learning rate yang disesuaikan sesuai kombinasi hyperparameter.

5. Pelatihan Model

Model dilatih menggunakan loop for melalui jumlah epoch yang ditentukan. Dalam setiap epoch:

- Gradien diatur ulang menggunakan optimizer.zero_grad.
- Prediksi dihasilkan dari model.
- Loss dihitung menggunakan criterion.
- Backpropagation dilakukan dengan loss.backward.
- Optimizer memperbarui parameter model dengan optimizer.step. Rata-rata loss dihitung setiap epoch, dan waktu pelatihan dicatat untuk memberikan wawasan tentang efisiensi.

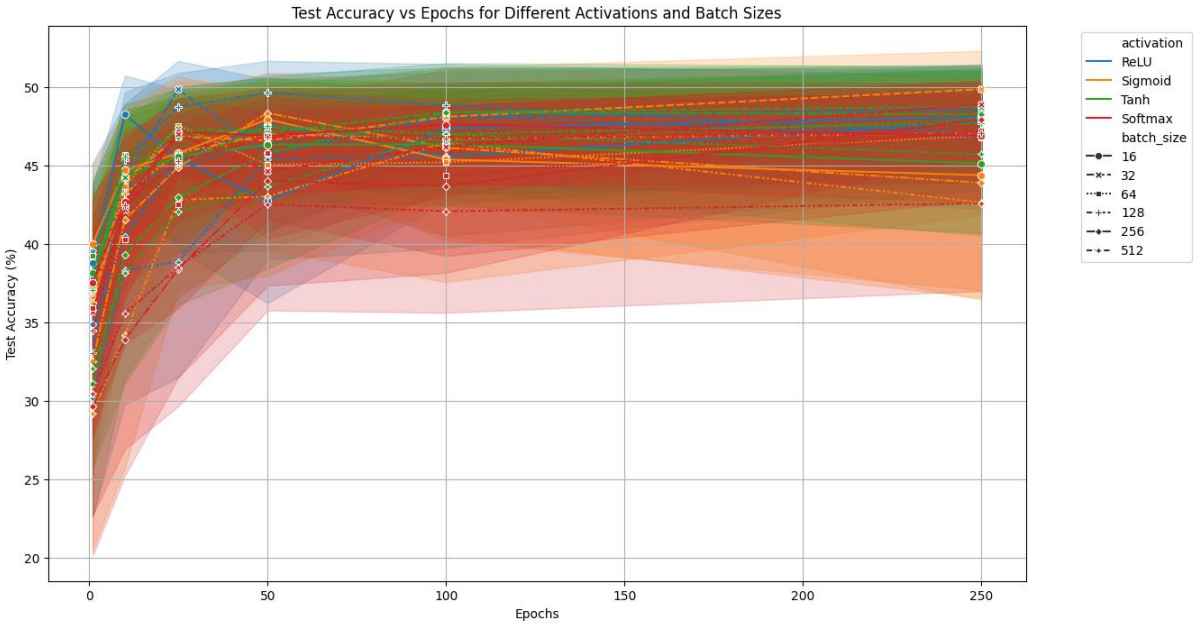
6. Evaluasi Model

Setelah pelatihan selesai, model dievaluasi menggunakan test_loader. Prediksi dibandingkan dengan label aktual untuk menghitung akurasi pengujian. Ini dilakukan tanpa menghitung gradien menggunakan torch.no_grad() untuk efisiensi.

7. Penyimpanan Hasil

Setiap kombinasi hyperparameter disimpan dalam results sebagai dictionary. Hasil mencakup semua konfigurasi hyperparameter, akurasi pengujian, dan lainnya.

Kode ini memberikan kerangka kerja yang sistematis untuk mengevaluasi berbagai konfigurasi model MLP dan mengidentifikasi kombinasi hyperparameter terbaik berdasarkan performa model.



Grafik ini menunjukkan hubungan antara akurasi pengujian (test accuracy) dan jumlah epoch untuk berbagai kombinasi fungsi aktivasi dan ukuran batch. Setiap garis dalam grafik merepresentasikan performa model untuk kombinasi fungsi aktivasi (ReLU, Sigmoid, Tanh, Softmax) dan ukuran batch tertentu (16, 32, 64, 128, 256, 512).

Pengamatan Utama:

1. **Tren Umum Akurasi**:
    - Akurasi meningkat secara signifikan selama epoch awal (0–50 epoch) untuk semua kombinasi, kemudian cenderung melambat atau mendatar setelah 100 epoch.
    - Model dengan batch size kecil (16, 32) umumnya menunjukkan akurasi lebih baik di awal pelatihan tetapi lebih fluktuatif dibandingkan batch size besar.
2. **Fungsi Aktivasi**:
    - Aktivasi Tanh dan ReLU menunjukkan performa yang lebih stabil dan cenderung mencapai akurasi tertinggi dibandingkan Sigmoid dan Softmax.
    - Aktivasi Sigmoid memiliki fluktuasi besar dalam beberapa kombinasi, menunjukkan ketidakstabilan dalam pelatihan.
3. **Efek Ukuran Batch**:
    - Batch size besar (128, 256, 512) menghasilkan pelatihan yang lebih stabil dengan fluktuasi kecil, tetapi cenderung membutuhkan lebih banyak epoch untuk mencapai akurasi optimal.
    - Batch size kecil mempercepat peningkatan akurasi awal tetapi dapat menyebabkan instabilitas pada akhir pelatihan.
4. **Stabilitas Pelatihan**:
    - Area bayangan di sekitar garis menunjukkan variabilitas akurasi antar percobaan. Kombinasi dengan area bayangan lebih kecil menunjukkan stabilitas yang lebih tinggi.