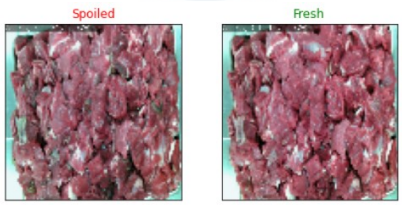
BAB 3
METODOLOGI PENELITIAN

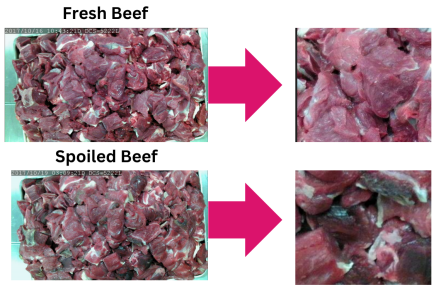
Metodologi penelitian ini dirancang untuk mengembangkan dan menguji model Convolutional Neural Network (CNN) dalam pengujian kesegaran daging sapi berdasarkan analisis visual citra. Langkah-langkah yang akan diambil adalah sebagai berikut:

3.1 Pengolahan Data

Pada tahap ini, data citra daging sapi diambil dari dataset Kaggle. Kemudian, gambar-gambar yang menggambarkan daging sapi yang sudah basi (Spoiled) dan yang masih segar (Fresh) akan dipotong dan diperbesar (zoom in) agar detailnya lebih jelas. Proses ini bertujuan untuk memastikan bahwa gambar-gambar yang akan digunakan dalam pelatihan model memiliki resolusi dan tingkat detail yang cukup untuk memungkinkan model untuk mempelajari perbedaan antara daging yang masih segar dan yang sudah basi dengan lebih baik. [19]



Gambar 3.1. Gambar Daging busuk dan daging segar



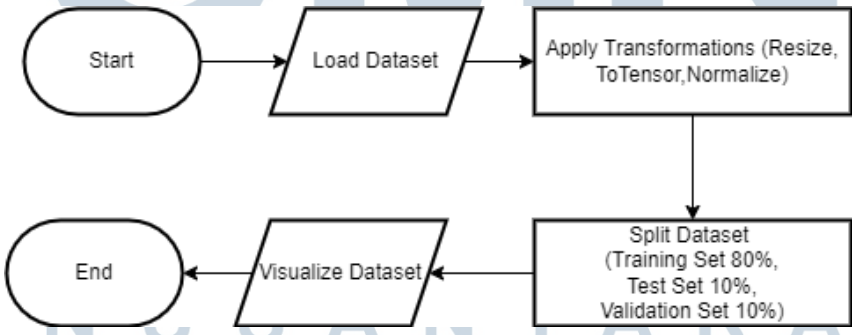
Gambar 3.2. Gambar Memotong dataset

3.2 Preprocessing Data

Preprocessing data dimulai dengan memuat dataset dari direktori yang ditentukan. Setelah dataset berhasil dimuat, dilakukan serangkaian transformasi untuk memastikan gambar berada dalam format yang tepat untuk pelatihan model. Transformasi pertama adalah mengubah ukuran gambar (resize) ke dimensi seragam, seperti 224x224 piksel, untuk menjaga konsistensi. Kemudian, gambar diubah menjadi tensor menggunakan metode ToTensor dari PyTorch, yang mengonversi gambar dari format PIL atau NumPy menjadi tensor. Langkah terakhir dalam transformasi adalah normalisasi (Normalize), yang menyesuaikan nilai piksel gambar ke rentang tertentu berdasarkan mean dan standar deviasi yang telah ditentukan, membantu mempercepat konvergensi selama pelatihan.

Setelah proses transformasi selesai, dataset dibagi menjadi tiga bagian: set pelatihan (training set), set validasi (validation set), dan set pengujian (test set). Pembagian dilakukan dengan rasio 80% untuk pelatihan, 10% untuk validasi, dan 10% untuk pengujian. Pembagian ini penting untuk memastikan bahwa model dapat dilatih dengan data yang cukup, divalidasi kinerjanya selama pelatihan, dan akhirnya diuji dengan data yang belum pernah dilihat untuk mengevaluasi generalisasi model. Pembagian dataset ini dilakukan menggunakan metode random_split dari PyTorch, yang secara acak membagi dataset berdasarkan rasio yang ditentukan. Hasil dari proses ini adalah tiga set data yang siap digunakan untuk melatih, memvalidasi, dan menguji model.

Dengan langkah-langkah Preprocessing ini, dataset gambar siap digunakan untuk proses pelatihan model ConvMixer, memastikan bahwa data berada dalam format yang optimal untuk mendapatkan hasil terbaik dari pelatihan model.



Gambar 3.3. Gambar flowchart preprocessing data

3.3 Penjelasan Arsitektur ConvMixer

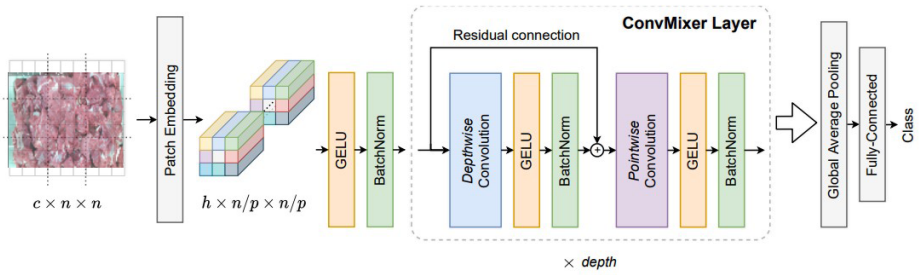
ConvMixer adalah sebuah arsitektur model Convolutional Neural Network (CNN) yang dirancang untuk mengekstrak fitur-fitur penting dari citra. Arsitektur ini terdiri dari beberapa komponen utama:

1. **Patch Embedding Layer:** Layer ini bertugas mengonversi input citra menjadi sejumlah patch (bagian-bagian kecil) dengan menggunakan sebuah lapisan konvolusi. Ini memungkinkan model untuk memperlakukan citra sebagai serangkaian patch untuk diproses lebih lanjut.

2. **ConvMixer Layer:** ConvMixer Layer adalah komponen utama dari arsitektur ConvMixer. Setiap ConvMixer Layer terdiri dari dua lapisan konvolusi: lapisan konvolusi kedalaman-w (depth-wise convolution) dan lapisan konvolusi titik-w (point-wise convolution). Lapisan-lapisan konvolusi ini bertujuan untuk mengekstrak fitur-fitur penting dari setiap patch citra.

3. **ConvMixer Architecture:** Arsitektur ConvMixer terdiri dari sejumlah ConvMixer Layers yang diurutkan secara berulang. Setiap ConvMixer Layer menerima input dari output ConvMixer Layer sebelumnya. Dengan cara ini, arsitektur ConvMixer dapat mengekstrak fitur-fitur hierarkis dari citra dengan berbagai tingkat kompleksitas.

4. **Adaptive Pooling and Classification Layer:** Pada akhir arsitektur ConvMixer, hasil keluaran dari ConvMixer Layer diolah dengan menggunakan lapisan adaptive average pooling untuk menghasilkan representasi fitur yang lebih kompak. Hasilnya kemudian digunakan oleh lapisan linear untuk klasifikasi citra ke dalam kelas-kelas yang diinginkan.



Gambar 3.4. Gambaran Output Model Convimxer

sumber :[14]

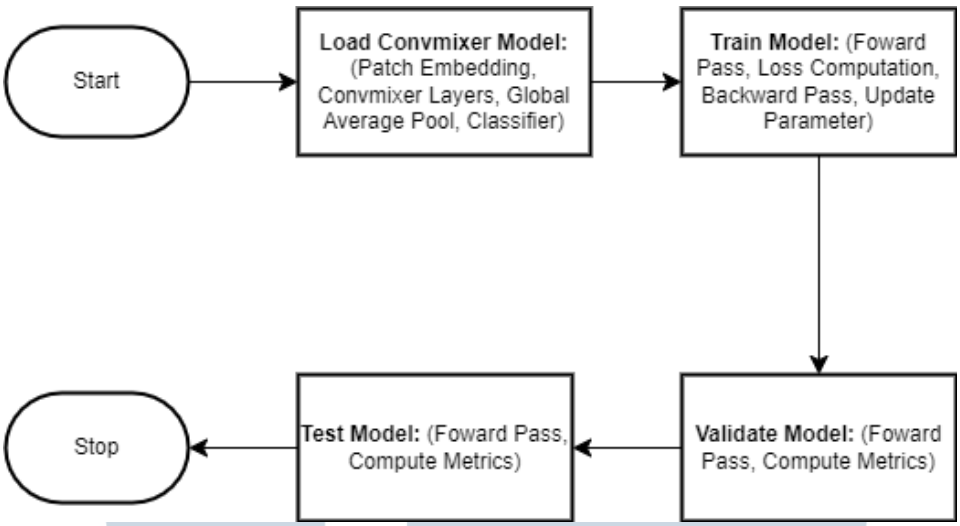
3.4 Pembuatan Model CNN ConvMixer

Layer	Output Shape	Param
Patch Embedding	(None, 128, 16, 16)	(196736)
ConvMixer Layers	(None, 128, 16, 16)	(204803)
Global Avg Pooling	(None, 128, 1, 1)	0
Flatten	(None, 128)	0
Classifier	(None, 3)	(387)
Total Param		(401926)
Trainable Param		(401926)
Non-trainable Param		0

Tabel 3.1. Tabel Layer model Convmixer

Selanjutnya, arsitektur ConvMixer didefinisikan, yang terdiri dari beberapa lapisan, termasuk lapisan Patch Embedding, lapisan ConvMixer, lapisan Global Average Pooling, dan lapisan Classifier. Patch Embedding bertanggung jawab untuk mengonversi gambar menjadi representasi yang sesuai untuk diproses oleh lapisan ConvMixer. ConvMixer Layers terdiri dari serangkaian lapisan ConvMixer, di mana setiap lapisan menerapkan operasi konvolusi dan normalisasi untuk mempelajari fitur-fitur penting dari gambar. Global Average Pooling digunakan untuk merata-ratakan fitur-fitur yang dihasilkan oleh lapisan ConvMixer menjadi vektor fitur tunggal. Terakhir, lapisan Classifier digunakan untuk mengklasifikasikan vektor fitur ke kelas-kelas yang sesuai.

Setelah arsitektur model ditentukan, proses pelatihan dimulai. Model dilatih dengan melalui beberapa iterasi (epoch), di mana setiap iterasi melibatkan langkah-langkah forward pass, perhitungan kerugian (loss), backward pass, dan pembaruan parameter menggunakan optimizer. Selama pelatihan, model juga dievaluasi menggunakan set validasi untuk memantau kinerjanya dan mencegah overfitting. Setelah pelatihan selesai, model diuji menggunakan set pengujian untuk mengevaluasi kinerjanya pada data yang belum pernah dilihat sebelumnya.



Gambar 3.5. Gambaran Flowchart Model Convmixer

3.5 Pelatihan Model ConvMixer

3.5.1 Inisialisasi Variabel

- `train_losses`, `val_losses`, `train_accuracies`, dan `val_accuracies` adalah *list* yang digunakan untuk menyimpan nilai *loss* dan *accuracy* selama proses pelatihan dan validasi.

3.5.2 Loop Epoch

- `for epoch in range(num_epochs):` Loop utama yang berjalan selama sejumlah *epoch* yang telah ditentukan (`num_epochs`).
- `model.train()`: Mengatur model ke mode *training*. Ini mempengaruhi beberapa *layer* seperti *Dropout* dan *BatchNorm* yang berperilaku berbeda selama *training* dan *testing*.
- `epoch_train_loss` diinisialisasi dengan nilai 0 untuk menyimpan total *loss* dalam satu *epoch*.

3.5.3 Progress Bar

- `loop = tqdm(enumerate(train_loader), total=len(train_loader), leave=False):` Membuat *progress bar* menggunakan `tqdm` untuk melacak proses *training* dalam satu *epoch*.

3.5.4 Loop Batch

- `for batch_index, (images, targets) in loop`: Loop ini berjalan untuk setiap *batch* data dalam `train_loader`.
- `images` dan `targets` dipindahkan ke perangkat yang sesuai (*device*), seperti GPU jika tersedia (`images.to(device)` dan `targets.to(device)`).

3.5.5 Mixed Precision Training

- `with torch.cuda.amp.autocast(enabled=device.type == 'cuda')`: Blok ini memungkinkan *mixed precision training* jika dijalankan di GPU, yang dapat mempercepat *training* dan mengurangi penggunaan memori.

3.5.6 Forward Pass dan Loss Calculation

- `logits = model(images)`: Melakukan *forward pass* melalui model dengan input `images`.
- `loss = criterion(logits, targets)`: Menghitung *loss* menggunakan fungsi *loss* yang telah ditentukan (`criterion`).

3.5.7 Backward Pass dan Optimizer Step

- `optimizer.zero_grad()`: Mengatur gradien ke nol sebelum melakukan *backpropagation* untuk mencegah akumulasi gradien dari iterasi sebelumnya.
- Jika `scaler` digunakan (untuk *mixed precision training*):
  - `scaler.scale(loss).backward()`: Melakukan *backpropagation* pada *loss* yang telah diskalakan.
  - `scaler.step(optimizer)`: Melakukan update pada parameter model menggunakan *optimizer*.
  - `scaler.update()`: Mengupdate `scaler` untuk menyesuaikan skala gradien di iterasi berikutnya.
- Jika `scaler` tidak digunakan:
  - `loss.backward()`: Melakukan *backpropagation* pada *loss*.

– `optimizer.step()`: Melakukan update pada parameter model menggunakan *optimizer*.

```python
class ConvMixer(nn.Module):
    def __init__(self, in_channels, dim, depth, kernel_size=9, patch_size=7, num_classes=2):
        super().__init__()
        self.patch_embedding = nn.Sequential(
            nn.Conv2d(in_channels, dim, kernel_size=patch_size, stride=patch_size),
            nn.GELU(),
            nn.BatchNorm2d(dim)
        )
        self.conv_mixer_layers = nn.ModuleList([ConvMixerLayer(dim, kernel_size) for _ in range(depth)])
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.flatten = nn.Flatten()
        self.classifier = nn.Linear(dim, num_classes)

    def forward(self, x):
        x = self.patch_embedding(x)
        for mixer in self.conv_mixer_layers:
            x = mixer(x)
        x = self.avgpool(x)
        x = self.flatten(x)
        return self.classifier(x)
```

Gambar 3.6. Gambaran proses pelatihan convmixer layer

3.5.8 Mengakumulasi dan Menampilkan Loss

- `epoch_train_loss += loss.item()`: Menambahkan nilai *loss* dari setiap *batch* ke total *loss* untuk *epoch* tersebut.
- `loop.set_description(f"Epoch [{epoch+1}/{num_epochs}] Loss: {loss.item():.4f}")`: Memperbarui deskripsi *progress bar* dengan menampilkan *epoch* saat ini dan nilai *loss* dari *batch* terakhir.

```python
# Define model, loss function, and optimizer
model = ConvMixer(in_channels, dim, depth, num_classes=num_classes).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=lr, weight_decay=1e-5)  # Add weight decay for L2 regularization

# Evaluation function
def evaluate(model, loader, criterion):
    model.eval()
    loss = 0.0
    correct_preds = 0
    total_preds = 0

    with torch.no_grad():
        for images, labels in loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            loss += criterion(outputs, labels).item() * images.size(0)
            _, predicted = torch.max(outputs, 1)
            correct_preds += (predicted == labels).sum().item()
            total_preds += labels.size(0)

    avg_loss = loss / len(loader.dataset)
    accuracy = correct_preds / total_preds * 100
    return avg_loss, accuracy
```

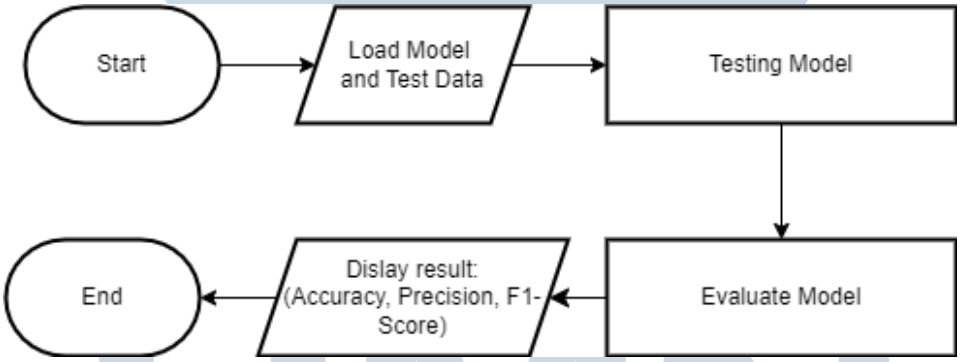
Gambar 3.7. Gambaran model loss

3.6 Validasi dan Evaluasi

Pada langkah ”Load Model”, model yang telah dilatih sebelumnya dimuat ke dalam memori. Selanjutnya, pada langkah ”Load Test Data”, data uji yang akan digunakan untuk mengevaluasi model dimuat ke dalam memori.

Setelah model dan data uji dimuat, langkah selanjutnya adalah ”Testing Model”, di mana model digunakan untuk melakukan prediksi pada data uji. Hasil prediksi ini kemudian digunakan pada langkah ”Evaluate Model”, di mana metrik evaluasi seperti akurasi, presisi, dan F1-Score dihitung berdasarkan perbandingan antara hasil prediksi dan label sebenarnya dari data uji.

Setelah metrik evaluasi dihitung, langkah terakhir adalah ”Display Result”, di mana hasil evaluasi seperti akurasi, presisi, dan F1-Score ditampilkan. Ini memungkinkan pengguna untuk melihat seberapa baik model melakukan pada data uji.



Gambar 3.8. Gambaran Flowchart Evaluasi

3.7 Spesifikasi Sistem

Dalam pengembangan sistem pengujian kesegaran daging menggunakan Convolutional Neural Network (CNN), berikut adalah spesifikasi sistem yang akan dipakai:

1. **Hardware**
   - Laptop: ROG GL553V
   - RAM: 16GB
   - Processor: Intel Core i7
2. **Software**

- Sistem Operasi: Windows 10
- Python: Python 3.x untuk menjalankan kode dan skrip.
- Text Editor: Google Colab Notebook

3. **Framework/ Library**
   - **Framework Deep Learning:**
     - **PyTorch**: Digunakan untuk pembuatan, pelatihan, dan evaluasi model Convolutional Neural Network (CNN).
   - **Library Pendukung:**
     - **NumPy**: Digunakan untuk manipulasi array dan operasi numerik. NumPy menyediakan struktur data array yang efisien dan berbagai fungsi matematis untuk operasi array.
     - **Pandas**: Digunakan untuk manipulasi dan analisis data. Pandas menyediakan struktur data seperti DataFrame yang memudahkan manipulasi data tabel.
     - **Matplotlib**: Digunakan untuk visualisasi data. Matplotlib menyediakan berbagai fungsi untuk membuat plot, grafik, dan visualisasi lainnya.
     - **scikit-learn**: Digunakan untuk evaluasi model dan metrik. scikit-learn menyediakan berbagai metrik evaluasi seperti akurasi, presisi, dan F1-Score, serta alat untuk pembelajaran mesin.
     - **tqdm**: Digunakan untuk menampilkan progress bar selama proses pelatihan. tqdm menyediakan antarmuka yang sederhana untuk menambahkan progress bar pada loop di Python.
   - **Library Pengolahan Citra:**
     - **OpenCV**: Digunakan untuk manipulasi citra dan augmentasi data. OpenCV menyediakan berbagai fungsi untuk pemrosesan citra, termasuk pembacaan, penulisan, dan transformasi citra.
   - **Virtual Environment:**
     - **Conda atau virtualenv**: Digunakan untuk mengisolasi lingkungan kerja. Penggunaan virtual environment membantu mengelola dependensi dan versi library sehingga lingkungan kerja menjadi konsisten dan terisolasi dari sistem utama.