Assignment 1 Deep Learning and Sensor fusion 2026

Multi class Classification

Necessary iomport here

In [1]:
```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Dataset
from torch.utils.data import Dataset
import torch.nn.functional as F
import matplotlib.pyplot as plt
from PIL import Image
from torchvision import transforms
import os
import numpy as np
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.model_selection import train_test_split

print(f"[INFO] Torch infos: {torch.__version__}")

DATASET_PATH = os.path.join("..", "Dataset", "BoneBreakClassification")
```

```
[INFO] Torch infos: 2.7.1+cu118
```

In [2]:
```
# some helpers
def load_data(dataset_path, class_names, type_of_data_to_load):
    # for each class name
    images, labels = list(), list()
    for el in class_names.keys():
        print(f"[WALKING] Walking into {el}")
        for images_name in os.listdir(os.path.join(dataset_path, el, type_of_data_t
            full_image_path = os.path.join(dataset_path, el, type_of_data_to_load,
            images.append(full_image_path)
            labels.append(class_names[el])
    return images, labels
```

set Dataset path

The path yopu see in the code point to my folder

```
.
├── Assignment_1
│   ├── Assignment_1_four_layers.ipynb
│   ├── Assignment_1_template.ipynb
│   ├── Assignment_1_three_layers.ipynb
│   └── cnn_ass_1.pt
├── Dataset
```

```
|   ├── BoneBreakClassification
```

If you put the data in a different place change it.

In [3]:
```
DATASET_PATH = os.path.join("..", "Dataset", "BoneBreakClassification")
```

(1) Load dataset

In [4]:
```
class_names = dict()

temp = os.listdir(DATASET_PATH)
temp = sorted(temp)

# prep up list and class number together
for i in range(len(temp)):
    class_names[temp[i]] = i

data_to_split, labels_to_split = load_data(dataset_path=DATASET_PATH,
                                           class_names=class_names,
                                           type_of_data_to_load="Train"

print(f"[DATA TO SPLIT] Data loaded complete {len(data_to_split)} Labeles are {len(

# load test data
test_data_handler, test_labels_handler = load_data(dataset_path=DATASET_PATH,
                                                   class_names=class_names,
                                                   type_of_data_to_load="Test")

print(f"[TRAIN DATA LOADED] Data loaded complete {len(test_data_handler)} Labeles a
```

```
[WALKING] Walking into Avulsion fracture
[WALKING] Walking into Comminuted fracture
[WALKING] Walking into Fracture Dislocation
[WALKING] Walking into Greenstick fracture
[WALKING] Walking into Hairline Fracture
[WALKING] Walking into Impacted fracture
[WALKING] Walking into Longitudinal fracture
[WALKING] Walking into Oblique fracture
[WALKING] Walking into Pathological fracture
[WALKING] Walking into Spiral Fracture
[DATA TO SPLIT] Data loaded complete 904 Labeles are 904
[WALKING] Walking into Avulsion fracture
[WALKING] Walking into Comminuted fracture
[WALKING] Walking into Fracture Dislocation
[WALKING] Walking into Greenstick fracture
[WALKING] Walking into Hairline Fracture
[WALKING] Walking into Impacted fracture
[WALKING] Walking into Longitudinal fracture
[WALKING] Walking into Oblique fracture
[WALKING] Walking into Pathological fracture
[WALKING] Walking into Spiral Fracture
[TRAIN DATA LOADED] Data loaded complete 131 Labeles are 131
```

(2) leave this cell because we need to create a val dataset and it is not there as default

In [5]:
```
# here you can split validation in stead of test so you can preserve intacted the t
train_data_handler, validation_data_handler, train_labels_handler, validation_label

print(f"[TRAIN DATA] The train data is {train_data_handler.shape}  and its labels a
print(f"[VALIDATION DATA] The validation data is {validation_data_handler.shape}  a

# my data loader need list yours? adapt it to your code
train_data_handler = list(train_data_handler)
train_labels_handler = list(train_labels_handler)

validation_data_handler = list(validation_data_handler)
validation_labels_handler = list(validation_labels_handler)
```

```
[TRAIN DATA] The train data is (723,)  and its labels are (723,)
[VALIDATION DATA] The validation data is (181,)  and its labels are (181,)
```

(3) Here ate two transforms one for training and one for evaluation

train transform need data augmentation:

- add random flip
- add random rotation
- add color jitter

both need need:

- resize
- toTensor handler
- Normalize (use the provided MEAN and STD)

Note that the evaluation transform does not need data augmentation.

In [6]:
```
TARGET_H = 224
TARGET_W = 224
IMAGE_MEAN = [0.485,0.456,0.406]
IMAGE_STD = [0.229,0.224,0.225]

# create transform
train_transform = transforms.Compose([
    transforms.Resize((TARGET_H, TARGET_W)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomVerticalFlip(),
    transforms.RandomRotation(degrees=15),
    transforms.ColorJitter(brightness=0.1, contrast=0.1),
```

```
    transforms.ToTensor(),
    transforms.Normalize(IMAGE_MEAN, IMAGE_STD),
])

evaluation_transform = transforms.Compose([
    transforms.Resize((TARGET_H, TARGET_W)),
    transforms.ToTensor(),
    transforms.Normalize(IMAGE_MEAN, IMAGE_STD),
])
```

In [7]:
```
class MyDataset(Dataset):
    def __init__(self, list_of_pathes, list_of_classes, transform=None) -> None:
        super().__init__()
        self.list_of_pathes = list_of_pathes
        self.list_of_classes = list_of_classes
        self.transform = transform

    def __len__(self):
        return len(self.list_of_pathes)

    def __getitem__(self, index):
        # select the image at index
        path = self.list_of_pathes[index]
        # select the label at index
        label = self.list_of_classes[index]
        # open the image Advice use Image package
        image = Image.open(path).convert("RGB")
        # apply transform
        if self.transform:
            image = self.transform(image)
        # return image and label
        return image, label
```

(4) Create data loder from the MyDataset class

In [8]:
```
train_dataset = MyDataset(list_of_pathes=train_data_handler,
                          list_of_classes=train_labels_handler,
                          transform=train_transform)

validation_dataset = MyDataset(list_of_pathes=validation_data_handler,
                               list_of_classes=validation_labels_handler,
                               transform=evaluation_transform)

test_dataset = MyDataset(list_of_pathes=test_data_handler,
                         list_of_classes=test_labels_handler,
                         transform=evaluation_transform)

# put those here
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True, num_workers=0
validation_loader = DataLoader(validation_dataset, batch_size=128, shuffle=False, n
test_loader = DataLoader(test_dataset, batch_size=128, shuffle=False, num_workers=0
```

Instance of the CNN

Use the Sequential API it is faster

- This cnn has two hidden layers (conv2d -> proper activation -> max pool 2d) one has 16 untis and the second has 32 units
- Add a flatten layer
- Classifier need 128 Linear unit proper activation and set dropdown at 0.5
- The last layer of the classifier how many units need?

In [9]:
```
class LayeredCNN(nn.Module):
    def __init__(self,
                 layers_list: list[int],
                 dense_units: list[int],
                 numbers_of_output_classes: int = 10,
                 kernel_size: int = 3,
                 padding_size: int = 1,
                 pooling_size: int = 2,
                 input_channels: int = 3,
                 dropout_probabilities: float = 0.5):
        super().__init__()

        layers = []
        input_units = input_channels

        for output_units in layers_list:

            layers += [
                nn.Conv2d(input_units, output_units, kernel_size=kernel_size, paddi
                nn.ReLU(inplace=True),
                nn.MaxPool2d(kernel_size=pooling_size)
            ]

            input_units = output_units

        self.features = nn.Sequential(*layers)

        multi_layer_perceptron = []

        in_features = dense_units[0]

        for h in dense_units[1:]:
            multi_layer_perceptron += [
                nn.Linear(in_features, h),
                nn.ReLU(inplace=True),
                nn.Dropout(p=dropout_probabilities),
            ]

            in_features = h

        self.classifier = nn.Sequential(*multi_layer_perceptron)
        self.flatten_layer = nn.Flatten()
        self.final_classifier = nn.Linear(in_features, numbers_of_output_classes)

    def forward(self, x):
        x = self.features(x)
```

```
        x = self.flatten_layer(x)
        x = self.classifier(x)
        x = self.final_classifier(x)

        return x
```

(5) Add proper optimizer and loss function set training epochs to 50

In [10]:
```
model = LayeredCNN([16,32], [100352, 128])

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model.to(device)

print(f"Model is running on: {device}")

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-3, weight_decay=1e-4)
```

```
Model is running on: cuda
```

In [11]:
```
def train_one_epoch(model, loader, criterion, optimizer, device):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0

    for images, labels in loader:
        images, labels = images.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # Track stats
        running_loss += loss.item() * images.size(0)
        predictions = outputs.argmax(dim=1)
        correct += (predictions == labels).sum().item()
        total += labels.size(0)

    return running_loss / total, correct / total
```

In [12]:
```
@torch.no_grad()

def evaluate(model, loader, criterion, device):
    model.eval()  # Turn off Dropout / BatchNorm
    running_loss = 0.0
    correct = 0
    total = 0

    for images, labels in loader:
        images, labels = images.to(device), labels.to(device)
```

```
        outputs = model(images)
        loss = criterion(outputs, labels)
        running_loss += loss.item() * images.size(0)
        predictions = outputs.argmax(dim=1)
        correct += (predictions == labels).sum().item()
        total += labels.size(0)

    return running_loss / total, correct / total
```

(6) add here the train and evaluation loop collect loss and accuracy, we want to see curves here

In [13]:
```
training_loss = []
validation_loss = []
training_accuracy = []
validation_accuracy = []

epochs = 50
best_validation_accuracy = 0.0

for epoch in range(1, epochs + 1):
    train_loss, train_accuracy = train_one_epoch(model = model, loader=train_loader
    val_loss, val_accuracy = evaluate(model=model, loader=validation_loader, criter

    training_loss.append(train_loss)
    validation_loss.append(val_loss)
    training_accuracy.append(train_accuracy)
    validation_accuracy.append(val_accuracy)

    print(f"[TRAINING EPOCH {epoch} / {epochs}] TRAIN LOSS {train_loss: .4f}, TRAIN


    if val_accuracy > best_validation_accuracy:
        best_validation_accuracy = val_accuracy
        torch.save(model.state_dict(), "cnn_bone_break_classification.pt")

print("TRAINING COMPLETE!!!")
print("BEST VALIDATION ACCURACY: ", best_validation_accuracy)
```

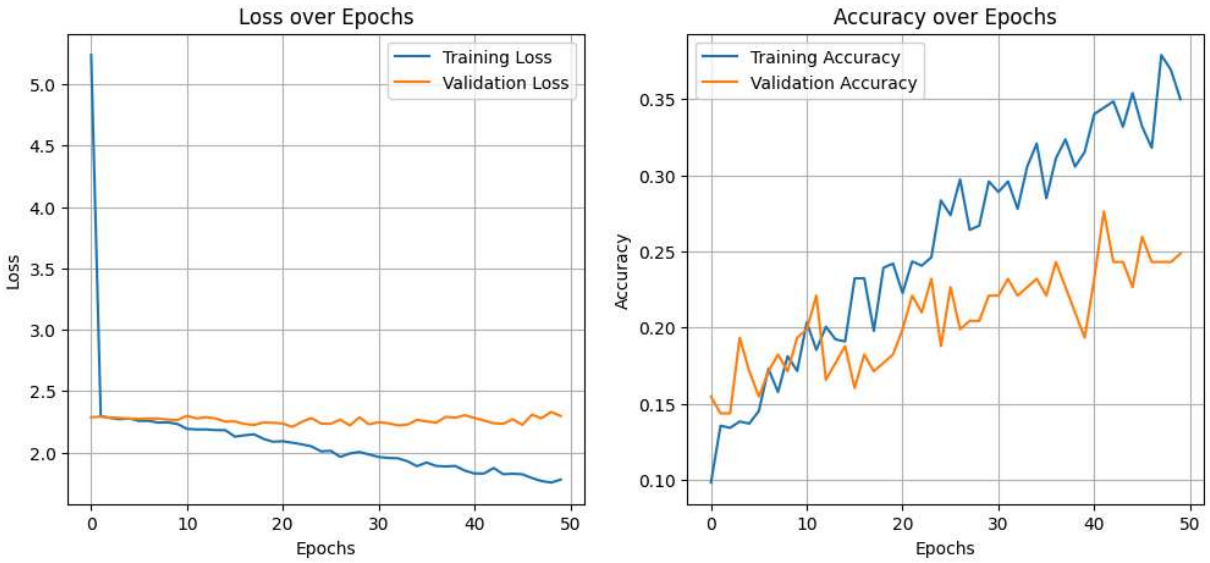
```
[TRAINING EPOCH 1 / 50] TRAIN LOSS  5.2374, TRAIN ACCURACY  0.098 | [VALIDATION EPOC
H 1 / 50] VALIDATION LOSS  2.2866, VALIDATION ACCURACY  0.155
[TRAINING EPOCH 2 / 50] TRAIN LOSS  2.2952, TRAIN ACCURACY  0.136 | [VALIDATION EPOC
H 2 / 50] VALIDATION LOSS  2.2912, VALIDATION ACCURACY  0.144
[TRAINING EPOCH 3 / 50] TRAIN LOSS  2.2804, TRAIN ACCURACY  0.134 | [VALIDATION EPOC
H 3 / 50] VALIDATION LOSS  2.2833, VALIDATION ACCURACY  0.144
[TRAINING EPOCH 4 / 50] TRAIN LOSS  2.2704, TRAIN ACCURACY  0.138 | [VALIDATION EPOC
H 4 / 50] VALIDATION LOSS  2.2806, VALIDATION ACCURACY  0.193
[TRAINING EPOCH 5 / 50] TRAIN LOSS  2.2762, TRAIN ACCURACY  0.137 | [VALIDATION EPOC
H 5 / 50] VALIDATION LOSS  2.2743, VALIDATION ACCURACY  0.171
[TRAINING EPOCH 6 / 50] TRAIN LOSS  2.2565, TRAIN ACCURACY  0.145 | [VALIDATION EPOC
H 6 / 50] VALIDATION LOSS  2.2728, VALIDATION ACCURACY  0.155
[TRAINING EPOCH 7 / 50] TRAIN LOSS  2.2570, TRAIN ACCURACY  0.173 | [VALIDATION EPOC
H 7 / 50] VALIDATION LOSS  2.2761, VALIDATION ACCURACY  0.171
[TRAINING EPOCH 8 / 50] TRAIN LOSS  2.2427, TRAIN ACCURACY  0.158 | [VALIDATION EPOC
H 8 / 50] VALIDATION LOSS  2.2753, VALIDATION ACCURACY  0.182
[TRAINING EPOCH 9 / 50] TRAIN LOSS  2.2443, TRAIN ACCURACY  0.181 | [VALIDATION EPOC
H 9 / 50] VALIDATION LOSS  2.2668, VALIDATION ACCURACY  0.171
[TRAINING EPOCH 10 / 50] TRAIN LOSS  2.2314, TRAIN ACCURACY  0.172 | [VALIDATION EPO
CH 10 / 50] VALIDATION LOSS  2.2631, VALIDATION ACCURACY  0.193
[TRAINING EPOCH 11 / 50] TRAIN LOSS  2.1917, TRAIN ACCURACY  0.203 | [VALIDATION EPO
CH 11 / 50] VALIDATION LOSS  2.2987, VALIDATION ACCURACY  0.199
[TRAINING EPOCH 12 / 50] TRAIN LOSS  2.1868, TRAIN ACCURACY  0.185 | [VALIDATION EPO
CH 12 / 50] VALIDATION LOSS  2.2761, VALIDATION ACCURACY  0.221
[TRAINING EPOCH 13 / 50] TRAIN LOSS  2.1870, TRAIN ACCURACY  0.201 | [VALIDATION EPO
CH 13 / 50] VALIDATION LOSS  2.2869, VALIDATION ACCURACY  0.166
[TRAINING EPOCH 14 / 50] TRAIN LOSS  2.1814, TRAIN ACCURACY  0.192 | [VALIDATION EPO
CH 14 / 50] VALIDATION LOSS  2.2764, VALIDATION ACCURACY  0.177
[TRAINING EPOCH 15 / 50] TRAIN LOSS  2.1811, TRAIN ACCURACY  0.191 | [VALIDATION EPO
CH 15 / 50] VALIDATION LOSS  2.2507, VALIDATION ACCURACY  0.188
[TRAINING EPOCH 16 / 50] TRAIN LOSS  2.1282, TRAIN ACCURACY  0.232 | [VALIDATION EPO
CH 16 / 50] VALIDATION LOSS  2.2520, VALIDATION ACCURACY  0.160
[TRAINING EPOCH 17 / 50] TRAIN LOSS  2.1394, TRAIN ACCURACY  0.232 | [VALIDATION EPO
CH 17 / 50] VALIDATION LOSS  2.2321, VALIDATION ACCURACY  0.182
[TRAINING EPOCH 18 / 50] TRAIN LOSS  2.1472, TRAIN ACCURACY  0.198 | [VALIDATION EPO
CH 18 / 50] VALIDATION LOSS  2.2235, VALIDATION ACCURACY  0.171
[TRAINING EPOCH 19 / 50] TRAIN LOSS  2.1096, TRAIN ACCURACY  0.239 | [VALIDATION EPO
CH 19 / 50] VALIDATION LOSS  2.2444, VALIDATION ACCURACY  0.177
[TRAINING EPOCH 20 / 50] TRAIN LOSS  2.0866, TRAIN ACCURACY  0.242 | [VALIDATION EPO
CH 20 / 50] VALIDATION LOSS  2.2408, VALIDATION ACCURACY  0.182
[TRAINING EPOCH 21 / 50] TRAIN LOSS  2.0906, TRAIN ACCURACY  0.223 | [VALIDATION EPO
CH 21 / 50] VALIDATION LOSS  2.2349, VALIDATION ACCURACY  0.199
[TRAINING EPOCH 22 / 50] TRAIN LOSS  2.0787, TRAIN ACCURACY  0.243 | [VALIDATION EPO
CH 22 / 50] VALIDATION LOSS  2.2079, VALIDATION ACCURACY  0.221
[TRAINING EPOCH 23 / 50] TRAIN LOSS  2.0653, TRAIN ACCURACY  0.241 | [VALIDATION EPO
CH 23 / 50] VALIDATION LOSS  2.2496, VALIDATION ACCURACY  0.210
[TRAINING EPOCH 24 / 50] TRAIN LOSS  2.0488, TRAIN ACCURACY  0.246 | [VALIDATION EPO
CH 24 / 50] VALIDATION LOSS  2.2801, VALIDATION ACCURACY  0.232
[TRAINING EPOCH 25 / 50] TRAIN LOSS  2.0079, TRAIN ACCURACY  0.284 | [VALIDATION EPO
CH 25 / 50] VALIDATION LOSS  2.2340, VALIDATION ACCURACY  0.188
[TRAINING EPOCH 26 / 50] TRAIN LOSS  2.0127, TRAIN ACCURACY  0.274 | [VALIDATION EPO
CH 26 / 50] VALIDATION LOSS  2.2333, VALIDATION ACCURACY  0.227
[TRAINING EPOCH 27 / 50] TRAIN LOSS  1.9634, TRAIN ACCURACY  0.297 | [VALIDATION EPO
CH 27 / 50] VALIDATION LOSS  2.2668, VALIDATION ACCURACY  0.199
[TRAINING EPOCH 28 / 50] TRAIN LOSS  1.9914, TRAIN ACCURACY  0.264 | [VALIDATION EPO
CH 28 / 50] VALIDATION LOSS  2.2202, VALIDATION ACCURACY  0.204
```

```
[TRAINING EPOCH 29 / 50] TRAIN LOSS  2.0037, TRAIN ACCURACY  0.267 | [VALIDATION EPO
CH 29 / 50] VALIDATION LOSS  2.2856, VALIDATION ACCURACY  0.204
[TRAINING EPOCH 30 / 50] TRAIN LOSS  1.9829, TRAIN ACCURACY  0.296 | [VALIDATION EPO
CH 30 / 50] VALIDATION LOSS  2.2283, VALIDATION ACCURACY  0.221
[TRAINING EPOCH 31 / 50] TRAIN LOSS  1.9624, TRAIN ACCURACY  0.289 | [VALIDATION EPO
CH 31 / 50] VALIDATION LOSS  2.2457, VALIDATION ACCURACY  0.221
[TRAINING EPOCH 32 / 50] TRAIN LOSS  1.9550, TRAIN ACCURACY  0.296 | [VALIDATION EPO
CH 32 / 50] VALIDATION LOSS  2.2378, VALIDATION ACCURACY  0.232
[TRAINING EPOCH 33 / 50] TRAIN LOSS  1.9519, TRAIN ACCURACY  0.278 | [VALIDATION EPO
CH 33 / 50] VALIDATION LOSS  2.2200, VALIDATION ACCURACY  0.221
[TRAINING EPOCH 34 / 50] TRAIN LOSS  1.9278, TRAIN ACCURACY  0.306 | [VALIDATION EPO
CH 34 / 50] VALIDATION LOSS  2.2262, VALIDATION ACCURACY  0.227
[TRAINING EPOCH 35 / 50] TRAIN LOSS  1.8877, TRAIN ACCURACY  0.321 | [VALIDATION EPO
CH 35 / 50] VALIDATION LOSS  2.2648, VALIDATION ACCURACY  0.232
[TRAINING EPOCH 36 / 50] TRAIN LOSS  1.9173, TRAIN ACCURACY  0.285 | [VALIDATION EPO
CH 36 / 50] VALIDATION LOSS  2.2527, VALIDATION ACCURACY  0.221
[TRAINING EPOCH 37 / 50] TRAIN LOSS  1.8894, TRAIN ACCURACY  0.311 | [VALIDATION EPO
CH 37 / 50] VALIDATION LOSS  2.2410, VALIDATION ACCURACY  0.243
[TRAINING EPOCH 38 / 50] TRAIN LOSS  1.8857, TRAIN ACCURACY  0.324 | [VALIDATION EPO
CH 38 / 50] VALIDATION LOSS  2.2894, VALIDATION ACCURACY  0.227
[TRAINING EPOCH 39 / 50] TRAIN LOSS  1.8893, TRAIN ACCURACY  0.306 | [VALIDATION EPO
CH 39 / 50] VALIDATION LOSS  2.2827, VALIDATION ACCURACY  0.210
[TRAINING EPOCH 40 / 50] TRAIN LOSS  1.8515, TRAIN ACCURACY  0.315 | [VALIDATION EPO
CH 40 / 50] VALIDATION LOSS  2.3031, VALIDATION ACCURACY  0.193
[TRAINING EPOCH 41 / 50] TRAIN LOSS  1.8276, TRAIN ACCURACY  0.340 | [VALIDATION EPO
CH 41 / 50] VALIDATION LOSS  2.2801, VALIDATION ACCURACY  0.232
[TRAINING EPOCH 42 / 50] TRAIN LOSS  1.8280, TRAIN ACCURACY  0.344 | [VALIDATION EPO
CH 42 / 50] VALIDATION LOSS  2.2596, VALIDATION ACCURACY  0.276
[TRAINING EPOCH 43 / 50] TRAIN LOSS  1.8726, TRAIN ACCURACY  0.349 | [VALIDATION EPO
CH 43 / 50] VALIDATION LOSS  2.2377, VALIDATION ACCURACY  0.243
[TRAINING EPOCH 44 / 50] TRAIN LOSS  1.8222, TRAIN ACCURACY  0.332 | [VALIDATION EPO
CH 44 / 50] VALIDATION LOSS  2.2328, VALIDATION ACCURACY  0.243
[TRAINING EPOCH 45 / 50] TRAIN LOSS  1.8262, TRAIN ACCURACY  0.354 | [VALIDATION EPO
CH 45 / 50] VALIDATION LOSS  2.2701, VALIDATION ACCURACY  0.227
[TRAINING EPOCH 46 / 50] TRAIN LOSS  1.8215, TRAIN ACCURACY  0.332 | [VALIDATION EPO
CH 46 / 50] VALIDATION LOSS  2.2242, VALIDATION ACCURACY  0.260
[TRAINING EPOCH 47 / 50] TRAIN LOSS  1.7908, TRAIN ACCURACY  0.318 | [VALIDATION EPO
CH 47 / 50] VALIDATION LOSS  2.3071, VALIDATION ACCURACY  0.243
[TRAINING EPOCH 48 / 50] TRAIN LOSS  1.7664, TRAIN ACCURACY  0.379 | [VALIDATION EPO
CH 48 / 50] VALIDATION LOSS  2.2781, VALIDATION ACCURACY  0.243
[TRAINING EPOCH 49 / 50] TRAIN LOSS  1.7541, TRAIN ACCURACY  0.369 | [VALIDATION EPO
CH 49 / 50] VALIDATION LOSS  2.3296, VALIDATION ACCURACY  0.243
[TRAINING EPOCH 50 / 50] TRAIN LOSS  1.7784, TRAIN ACCURACY  0.350 | [VALIDATION EPO
CH 50 / 50] VALIDATION LOSS  2.2969, VALIDATION ACCURACY  0.249
TRAINING COMPLETE!!!
BEST VALIDATION ACCURACY:  0.27624309392265195
```

In [14]:

```python
plt.figure(figsize=(12, 5))

# Plot 1: Loss Curve
plt.subplot(1, 2, 1)
plt.plot(training_loss, label='Training Loss')
plt.plot(validation_loss, label='Validation Loss')
plt.title('Loss over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
```

```
plt.legend()
plt.grid(True)

# Plot 2: Accuracy Curve
plt.subplot(1, 2, 2)
plt.plot(training_accuracy, label='Training Accuracy')
plt.plot(validation_accuracy, label='Validation Accuracy')
plt.title('Accuracy over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)

plt.show()
```

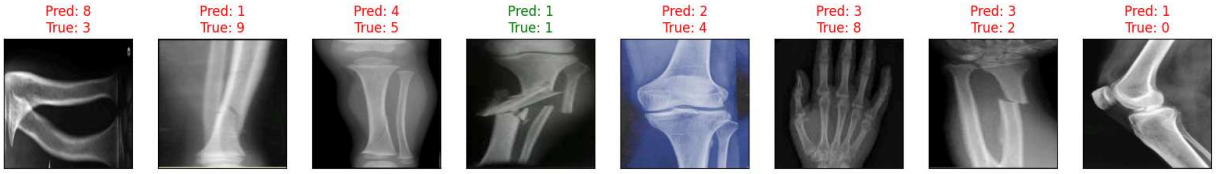


(7) Make prediction and shows some predicted images

In [15]:
```
def display_image(img):
    img = img.cpu().numpy().transpose((1, 2, 0))
    mean = np.array([0.485, 0.456, 0.406])
    std = np.array([0.229, 0.224, 0.225])
    img = std * img + mean
    img = np.clip(img, 0, 1)
    plt.imshow(img)


model.eval()

iterator = iter(validation_loader)

images, labels = next(iterator)

images, labels = images.to(device), labels.to(device)

print(f"Got a batch of {images.size(0)} images.")
```

```
print(f"Image shape: {images.shape}")

with torch.no_grad():
    outputs = model(images)
    _, preds = torch.max(outputs, 1)
fig = plt.figure(figsize=(25, 4))

for idx in range(8):

    ax = fig.add_subplot(1, 10, idx+1, xticks=[], yticks=[])

    display_image(images[idx])


    pred_label = preds[idx].item()
    truth_label = labels[idx].item()

    color = "green" if pred_label == truth_label else "red"

    ax.set_title(f"Pred: {pred_label}\nTrue: {truth_label}", color=color,)

plt.show()
```

```
Got a batch of 128 images.
Image shape: torch.Size([128, 3, 224, 224])
```



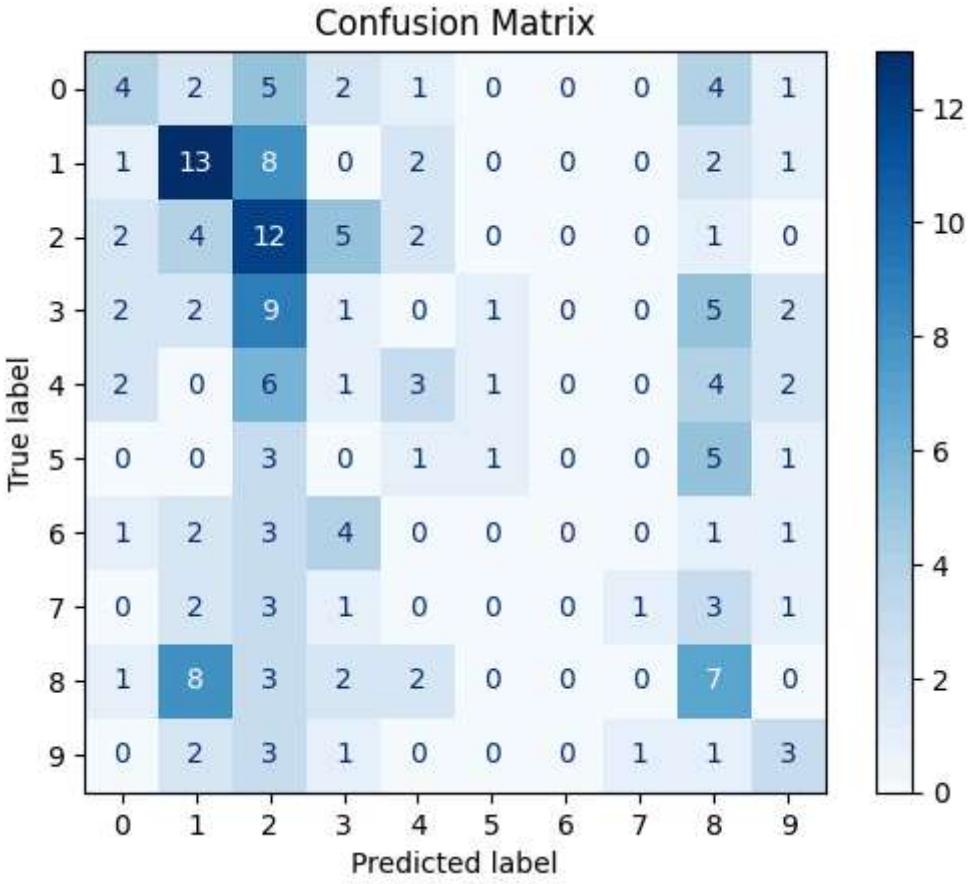
(8) Make Evaluation with confusion matrix

In [16]:

```
all_predictions = []
all_labels = []

model.eval()

with torch.no_grad():
    for images, labels in validation_loader:
        images = images.to(device)
        labels = labels.to(device)

        outputs = model(images)
        _, preds = torch.max(outputs, 1)

        all_predictions.extend(preds.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())

cm = confusion_matrix(all_labels, all_predictions)

plt.figure(figsize=(20, 20))

disp = ConfusionMatrixDisplay(confusion_matrix=cm)
```

```
disp.plot(cmap='Blues', values_format='d')

plt.title("Confusion Matrix")
plt.show()
```

```
<Figure size 2000x2000 with 0 Axes>
```



(9) Instance of a new CNN

Use the Sequential API it is faster

- This cnn has tre hidden layers (conv2d -> proper activation -> max pool 2d) one has 16 untis, the second has 32 units, and the third has 64
- Add a flatten layer
- Classifier need 128 Linear unit proper activation and set dropdown at 0.5
- The last layer of the classifier how many units need?

Repeat from 5 to 8

In [17]:
```
model3 = LayeredCNN([16,32, 64], [50176, 128])

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model3.to(device)
```

```
print(f"Model3 is running on: {device}")

criterion = nn.CrossEntropyLoss()

optimizer3 = torch.optim.Adam(model3.parameters(), lr=1e-3, weight_decay=1e-4)
```

```
Model3 is running on: cuda
```

In [18]:
```
training_loss_3 = []
validation_loss_3 = []
training_accuracy_3 = []
validation_accuracy_3 = []

epochs = 50
best_validation_accuracy_3 = 0.0

for epoch in range(1, epochs + 1):

    train_loss, train_accuracy = train_one_epoch(
        model=model3,
        loader=train_loader,
        criterion=criterion,
        optimizer=optimizer3,
        device=device
    )


    val_loss, val_accuracy = evaluate(
        model=model3,
        loader=validation_loader,
        criterion=criterion,
        device=device
    )

    # Append to the new lists
    training_loss_3.append(train_loss)
    validation_loss_3.append(val_loss)
    training_accuracy_3.append(train_accuracy)
    validation_accuracy_3.append(val_accuracy)

    print(f"[TRAINING EPOCH {epoch} / {epochs}] TRAIN LOSS {train_loss: .4f}, TRAIN

    if val_accuracy > best_validation_accuracy_3:
        best_validation_accuracy_3 = val_accuracy
        torch.save(model3.state_dict(), "cnn_3layer_bone_break_classification.pt")

print("TRAINING COMPLETE!!!")
print("BEST VALIDATION ACCURACY (3 Layer): ", best_validation_accuracy_3)
```

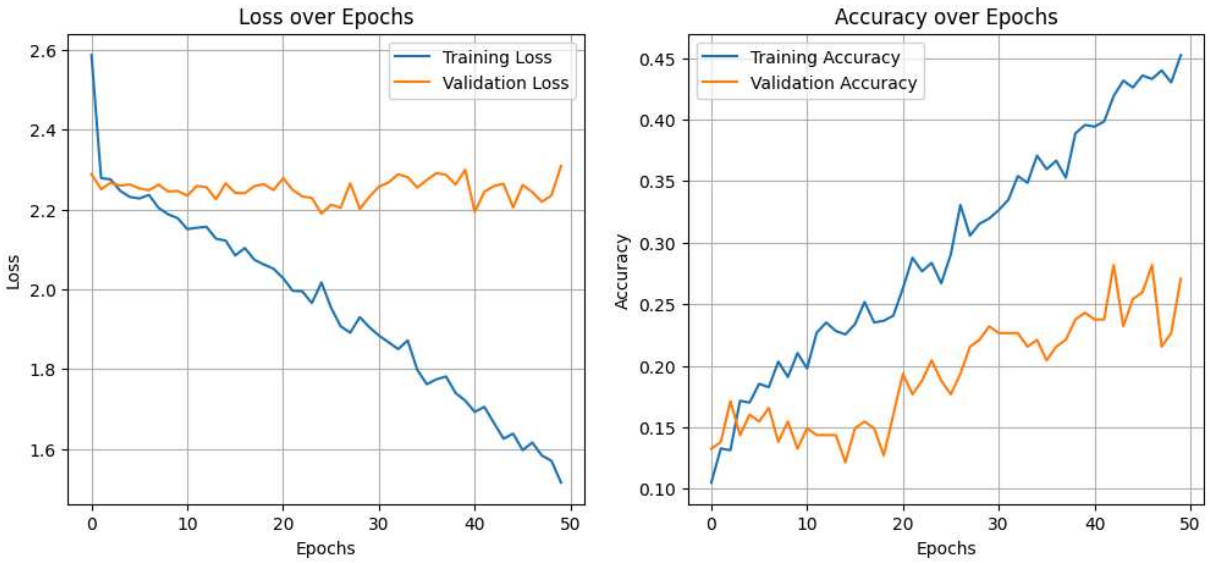
```
[TRAINING EPOCH 1 / 50] TRAIN LOSS  2.5866, TRAIN ACCURACY  0.105 | [VALIDATION EPOC
H 1 / 50] VALIDATION LOSS  2.2885, VALIDATION ACCURACY  0.133
[TRAINING EPOCH 2 / 50] TRAIN LOSS  2.2786, TRAIN ACCURACY  0.133 | [VALIDATION EPOC
H 2 / 50] VALIDATION LOSS  2.2507, VALIDATION ACCURACY  0.138
[TRAINING EPOCH 3 / 50] TRAIN LOSS  2.2752, TRAIN ACCURACY  0.131 | [VALIDATION EPOC
H 3 / 50] VALIDATION LOSS  2.2668, VALIDATION ACCURACY  0.171
[TRAINING EPOCH 4 / 50] TRAIN LOSS  2.2464, TRAIN ACCURACY  0.172 | [VALIDATION EPOC
H 4 / 50] VALIDATION LOSS  2.2596, VALIDATION ACCURACY  0.144
[TRAINING EPOCH 5 / 50] TRAIN LOSS  2.2311, TRAIN ACCURACY  0.170 | [VALIDATION EPOC
H 5 / 50] VALIDATION LOSS  2.2624, VALIDATION ACCURACY  0.160
[TRAINING EPOCH 6 / 50] TRAIN LOSS  2.2274, TRAIN ACCURACY  0.185 | [VALIDATION EPOC
H 6 / 50] VALIDATION LOSS  2.2527, VALIDATION ACCURACY  0.155
[TRAINING EPOCH 7 / 50] TRAIN LOSS  2.2363, TRAIN ACCURACY  0.183 | [VALIDATION EPOC
H 7 / 50] VALIDATION LOSS  2.2479, VALIDATION ACCURACY  0.166
[TRAINING EPOCH 8 / 50] TRAIN LOSS  2.2039, TRAIN ACCURACY  0.203 | [VALIDATION EPOC
H 8 / 50] VALIDATION LOSS  2.2622, VALIDATION ACCURACY  0.138
[TRAINING EPOCH 9 / 50] TRAIN LOSS  2.1876, TRAIN ACCURACY  0.191 | [VALIDATION EPOC
H 9 / 50] VALIDATION LOSS  2.2447, VALIDATION ACCURACY  0.155
[TRAINING EPOCH 10 / 50] TRAIN LOSS  2.1779, TRAIN ACCURACY  0.210 | [VALIDATION EPO
CH 10 / 50] VALIDATION LOSS  2.2462, VALIDATION ACCURACY  0.133
[TRAINING EPOCH 11 / 50] TRAIN LOSS  2.1505, TRAIN ACCURACY  0.198 | [VALIDATION EPO
CH 11 / 50] VALIDATION LOSS  2.2345, VALIDATION ACCURACY  0.149
[TRAINING EPOCH 12 / 50] TRAIN LOSS  2.1540, TRAIN ACCURACY  0.227 | [VALIDATION EPO
CH 12 / 50] VALIDATION LOSS  2.2588, VALIDATION ACCURACY  0.144
[TRAINING EPOCH 13 / 50] TRAIN LOSS  2.1562, TRAIN ACCURACY  0.235 | [VALIDATION EPO
CH 13 / 50] VALIDATION LOSS  2.2554, VALIDATION ACCURACY  0.144
[TRAINING EPOCH 14 / 50] TRAIN LOSS  2.1264, TRAIN ACCURACY  0.228 | [VALIDATION EPO
CH 14 / 50] VALIDATION LOSS  2.2255, VALIDATION ACCURACY  0.144
[TRAINING EPOCH 15 / 50] TRAIN LOSS  2.1217, TRAIN ACCURACY  0.225 | [VALIDATION EPO
CH 15 / 50] VALIDATION LOSS  2.2655, VALIDATION ACCURACY  0.122
[TRAINING EPOCH 16 / 50] TRAIN LOSS  2.0846, TRAIN ACCURACY  0.234 | [VALIDATION EPO
CH 16 / 50] VALIDATION LOSS  2.2414, VALIDATION ACCURACY  0.149
[TRAINING EPOCH 17 / 50] TRAIN LOSS  2.1032, TRAIN ACCURACY  0.252 | [VALIDATION EPO
CH 17 / 50] VALIDATION LOSS  2.2406, VALIDATION ACCURACY  0.155
[TRAINING EPOCH 18 / 50] TRAIN LOSS  2.0739, TRAIN ACCURACY  0.235 | [VALIDATION EPO
CH 18 / 50] VALIDATION LOSS  2.2584, VALIDATION ACCURACY  0.149
[TRAINING EPOCH 19 / 50] TRAIN LOSS  2.0618, TRAIN ACCURACY  0.237 | [VALIDATION EPO
CH 19 / 50] VALIDATION LOSS  2.2630, VALIDATION ACCURACY  0.127
[TRAINING EPOCH 20 / 50] TRAIN LOSS  2.0513, TRAIN ACCURACY  0.241 | [VALIDATION EPO
CH 20 / 50] VALIDATION LOSS  2.2485, VALIDATION ACCURACY  0.160
[TRAINING EPOCH 21 / 50] TRAIN LOSS  2.0276, TRAIN ACCURACY  0.263 | [VALIDATION EPO
CH 21 / 50] VALIDATION LOSS  2.2784, VALIDATION ACCURACY  0.193
[TRAINING EPOCH 22 / 50] TRAIN LOSS  1.9963, TRAIN ACCURACY  0.288 | [VALIDATION EPO
CH 22 / 50] VALIDATION LOSS  2.2487, VALIDATION ACCURACY  0.177
[TRAINING EPOCH 23 / 50] TRAIN LOSS  1.9946, TRAIN ACCURACY  0.277 | [VALIDATION EPO
CH 23 / 50] VALIDATION LOSS  2.2324, VALIDATION ACCURACY  0.188
[TRAINING EPOCH 24 / 50] TRAIN LOSS  1.9654, TRAIN ACCURACY  0.284 | [VALIDATION EPO
CH 24 / 50] VALIDATION LOSS  2.2289, VALIDATION ACCURACY  0.204
[TRAINING EPOCH 25 / 50] TRAIN LOSS  2.0170, TRAIN ACCURACY  0.267 | [VALIDATION EPO
CH 25 / 50] VALIDATION LOSS  2.1893, VALIDATION ACCURACY  0.188
[TRAINING EPOCH 26 / 50] TRAIN LOSS  1.9539, TRAIN ACCURACY  0.290 | [VALIDATION EPO
CH 26 / 50] VALIDATION LOSS  2.2116, VALIDATION ACCURACY  0.177
[TRAINING EPOCH 27 / 50] TRAIN LOSS  1.9071, TRAIN ACCURACY  0.331 | [VALIDATION EPO
CH 27 / 50] VALIDATION LOSS  2.2042, VALIDATION ACCURACY  0.193
[TRAINING EPOCH 28 / 50] TRAIN LOSS  1.8910, TRAIN ACCURACY  0.306 | [VALIDATION EPO
CH 28 / 50] VALIDATION LOSS  2.2653, VALIDATION ACCURACY  0.215
```

```
[TRAINING EPOCH 29 / 50] TRAIN LOSS  1.9299, TRAIN ACCURACY  0.315 | [VALIDATION EPO
CH 29 / 50] VALIDATION LOSS  2.2013, VALIDATION ACCURACY  0.221
[TRAINING EPOCH 30 / 50] TRAIN LOSS  1.9048, TRAIN ACCURACY  0.320 | [VALIDATION EPO
CH 30 / 50] VALIDATION LOSS  2.2313, VALIDATION ACCURACY  0.232
[TRAINING EPOCH 31 / 50] TRAIN LOSS  1.8840, TRAIN ACCURACY  0.326 | [VALIDATION EPO
CH 31 / 50] VALIDATION LOSS  2.2569, VALIDATION ACCURACY  0.227
[TRAINING EPOCH 32 / 50] TRAIN LOSS  1.8671, TRAIN ACCURACY  0.335 | [VALIDATION EPO
CH 32 / 50] VALIDATION LOSS  2.2673, VALIDATION ACCURACY  0.227
[TRAINING EPOCH 33 / 50] TRAIN LOSS  1.8499, TRAIN ACCURACY  0.354 | [VALIDATION EPO
CH 33 / 50] VALIDATION LOSS  2.2880, VALIDATION ACCURACY  0.227
[TRAINING EPOCH 34 / 50] TRAIN LOSS  1.8715, TRAIN ACCURACY  0.349 | [VALIDATION EPO
CH 34 / 50] VALIDATION LOSS  2.2808, VALIDATION ACCURACY  0.215
[TRAINING EPOCH 35 / 50] TRAIN LOSS  1.7980, TRAIN ACCURACY  0.371 | [VALIDATION EPO
CH 35 / 50] VALIDATION LOSS  2.2546, VALIDATION ACCURACY  0.221
[TRAINING EPOCH 36 / 50] TRAIN LOSS  1.7620, TRAIN ACCURACY  0.360 | [VALIDATION EPO
CH 36 / 50] VALIDATION LOSS  2.2737, VALIDATION ACCURACY  0.204
[TRAINING EPOCH 37 / 50] TRAIN LOSS  1.7742, TRAIN ACCURACY  0.367 | [VALIDATION EPO
CH 37 / 50] VALIDATION LOSS  2.2907, VALIDATION ACCURACY  0.215
[TRAINING EPOCH 38 / 50] TRAIN LOSS  1.7814, TRAIN ACCURACY  0.353 | [VALIDATION EPO
CH 38 / 50] VALIDATION LOSS  2.2866, VALIDATION ACCURACY  0.221
[TRAINING EPOCH 39 / 50] TRAIN LOSS  1.7398, TRAIN ACCURACY  0.389 | [VALIDATION EPO
CH 39 / 50] VALIDATION LOSS  2.2621, VALIDATION ACCURACY  0.238
[TRAINING EPOCH 40 / 50] TRAIN LOSS  1.7210, TRAIN ACCURACY  0.396 | [VALIDATION EPO
CH 40 / 50] VALIDATION LOSS  2.2995, VALIDATION ACCURACY  0.243
[TRAINING EPOCH 41 / 50] TRAIN LOSS  1.6927, TRAIN ACCURACY  0.394 | [VALIDATION EPO
CH 41 / 50] VALIDATION LOSS  2.1934, VALIDATION ACCURACY  0.238
[TRAINING EPOCH 42 / 50] TRAIN LOSS  1.7052, TRAIN ACCURACY  0.398 | [VALIDATION EPO
CH 42 / 50] VALIDATION LOSS  2.2440, VALIDATION ACCURACY  0.238
[TRAINING EPOCH 43 / 50] TRAIN LOSS  1.6650, TRAIN ACCURACY  0.419 | [VALIDATION EPO
CH 43 / 50] VALIDATION LOSS  2.2588, VALIDATION ACCURACY  0.282
[TRAINING EPOCH 44 / 50] TRAIN LOSS  1.6255, TRAIN ACCURACY  0.432 | [VALIDATION EPO
CH 44 / 50] VALIDATION LOSS  2.2642, VALIDATION ACCURACY  0.232
[TRAINING EPOCH 45 / 50] TRAIN LOSS  1.6381, TRAIN ACCURACY  0.426 | [VALIDATION EPO
CH 45 / 50] VALIDATION LOSS  2.2048, VALIDATION ACCURACY  0.254
[TRAINING EPOCH 46 / 50] TRAIN LOSS  1.5967, TRAIN ACCURACY  0.436 | [VALIDATION EPO
CH 46 / 50] VALIDATION LOSS  2.2611, VALIDATION ACCURACY  0.260
[TRAINING EPOCH 47 / 50] TRAIN LOSS  1.6158, TRAIN ACCURACY  0.433 | [VALIDATION EPO
CH 47 / 50] VALIDATION LOSS  2.2436, VALIDATION ACCURACY  0.282
[TRAINING EPOCH 48 / 50] TRAIN LOSS  1.5833, TRAIN ACCURACY  0.440 | [VALIDATION EPO
CH 48 / 50] VALIDATION LOSS  2.2190, VALIDATION ACCURACY  0.215
[TRAINING EPOCH 49 / 50] TRAIN LOSS  1.5701, TRAIN ACCURACY  0.430 | [VALIDATION EPO
CH 49 / 50] VALIDATION LOSS  2.2348, VALIDATION ACCURACY  0.227
[TRAINING EPOCH 50 / 50] TRAIN LOSS  1.5156, TRAIN ACCURACY  0.452 | [VALIDATION EPO
CH 50 / 50] VALIDATION LOSS  2.3087, VALIDATION ACCURACY  0.271
TRAINING COMPLETE!!!
BEST VALIDATION ACCURACY (3 Layer):  0.281767955801105
```

In [19]:

```python
plt.figure(figsize=(12, 5))

# Plot 1: Loss Curve
plt.subplot(1, 2, 1)
plt.plot(training_loss_3, label='Training Loss')
plt.plot(validation_loss_3, label='Validation Loss')
plt.title('Loss over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
```

```
plt.legend()
plt.grid(True)

# Plot 2: Accuracy Curve
plt.subplot(1, 2, 2)
plt.plot(training_accuracy_3, label='Training Accuracy')
plt.plot(validation_accuracy_3, label='Validation Accuracy')
plt.title('Accuracy over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)

plt.show()
```

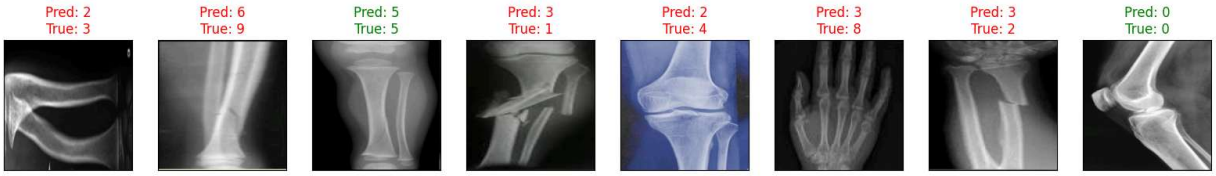


In [20]:

```
def display_image(img):
    img = img.cpu().numpy().transpose((1, 2, 0))
    mean = np.array([0.485, 0.456, 0.406])
    std = np.array([0.229, 0.224, 0.225])
    img = std * img + mean
    img = np.clip(img, 0, 1)
    plt.imshow(img)

model3.eval()

iterator = iter(validation_loader)

images, labels = next(iterator)

images, labels = images.to(device), labels.to(device)

print(f"Got a batch of {images.size(0)} images.")
print(f"Image shape: {images.shape}")

with torch.no_grad():

    outputs = model3(images)
    _, preds = torch.max(outputs, 1)
```

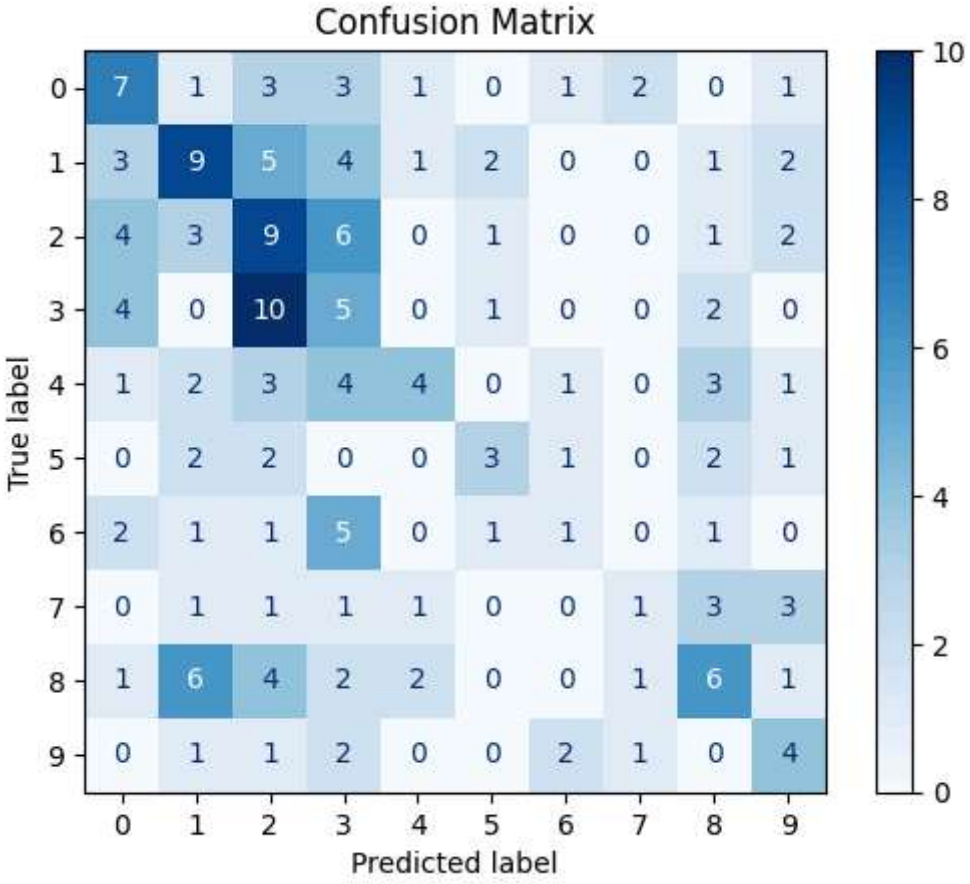
```
fig = plt.figure(figsize=(25, 4))

for idx in range(8):

    ax = fig.add_subplot(1, 10, idx+1, xticks=[], yticks=[])

    display_image(images[idx])


    pred_label = preds[idx].item()
    truth_label = labels[idx].item()

    color = "green" if pred_label == truth_label else "red"

    ax.set_title(f"Pred: {pred_label}\nTrue: {truth_label}", color=color,)

plt.show()
```

```
Got a batch of 128 images.
Image shape: torch.Size([128, 3, 224, 224])
```



In [21]:
```
all_predictions = []
all_labels = []

model3.eval()

with torch.no_grad():
    for images, labels in validation_loader:
        images = images.to(device)
        labels = labels.to(device)

        outputs = model3(images)
        _, preds = torch.max(outputs, 1)

        all_predictions.extend(preds.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())

cm = confusion_matrix(all_labels, all_predictions)

plt.figure(figsize=(20, 20))

disp = ConfusionMatrixDisplay(confusion_matrix=cm)

disp.plot(cmap='Blues', values_format='d')

plt.title("Confusion Matrix")
plt.show()
```

```
<Figure size 2000x2000 with 0 Axes>
```


Confusion Matrix
True label
Predicted label
0 7 1 3 3 1 0 1 2 0 1
1 3 9 5 4 1 2 0 0 1 2
2 4 3 9 6 0 1 0 0 1 2
3 4 0 10 5 0 1 0 0 2 0
4 1 2 3 4 4 0 1 0 3 1
5 0 2 2 0 0 3 1 0 2 1
6 2 1 1 5 0 1 1 0 1 0
7 0 1 1 1 1 0 0 1 3 3
8 1 6 4 2 2 0 0 1 6 1
9 0 1 1 2 0 0 2 1 0 4
0 1 2 3 4 5 6 7 8 9
10 8 6 4 2 0