Bone Fracture Detection with 95% Model Compression using Graphon NTK Theory!

```
import numpy as np
import pandas as pd
import os

base_path = "/kaggle/input/fracture-multi-region-x-ray-
data/Bone_Fracture_Binary_Classification/Bone_Fracture_Binary_Classification/
train/"
categories = ["fractured", "not fractured"]

image_paths = []
labels = []

for category in categories:
    category_path = os.path.join(base_path, category)
    for image_name in os.listdir(category_path):
        image_path = os.path.join(category_path, image_name)
        image_paths.append(image_path)
        labels.append(category)

df = pd.DataFrame({
    "image_path": image_paths,
    "label": labels
})

df.head()
```

	image_path	label
0	/kaggle/input/fracture-multi-region-x-ray-data...	fractured
1	/kaggle/input/fracture-multi-region-x-ray-data...	fractured
2	/kaggle/input/fracture-multi-region-x-ray-data...	fractured

```
3  /kaggle/input/fracture-multi-region-x-ray-data...  fractured
4  /kaggle/input/fracture-multi-region-x-ray-data...  fractured
```

```
df.tail()
```

```
                                           image_path          label
9241  /kaggle/input/fracture-multi-region-x-ray-data...  not fractured
9242  /kaggle/input/fracture-multi-region-x-ray-data...  not fractured
9243  /kaggle/input/fracture-multi-region-x-ray-data...  not fractured
9244  /kaggle/input/fracture-multi-region-x-ray-data...  not fractured
9245  /kaggle/input/fracture-multi-region-x-ray-data...  not fractured
```

```
df.shape
```

```
(9246, 2)
```

```
df.columns
```

```
Index(['image_path', 'label'], dtype='object')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9246 entries, 0 to 9245
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   image_path  9246 non-null   object
 1   label       9246 non-null   object
dtypes: object(2)
memory usage: 144.6+ KB
```

```
df['label'].unique()
```

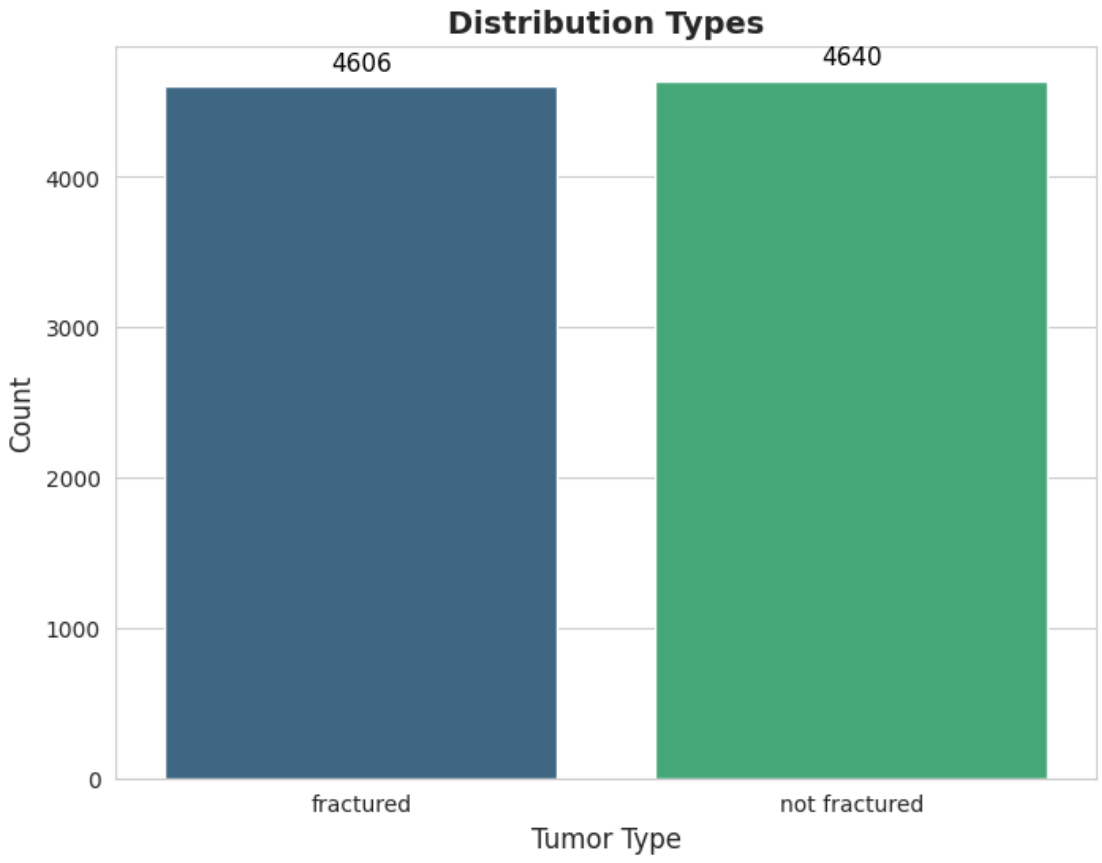
```
array(['fractured', 'not fractured'], dtype=object)
```

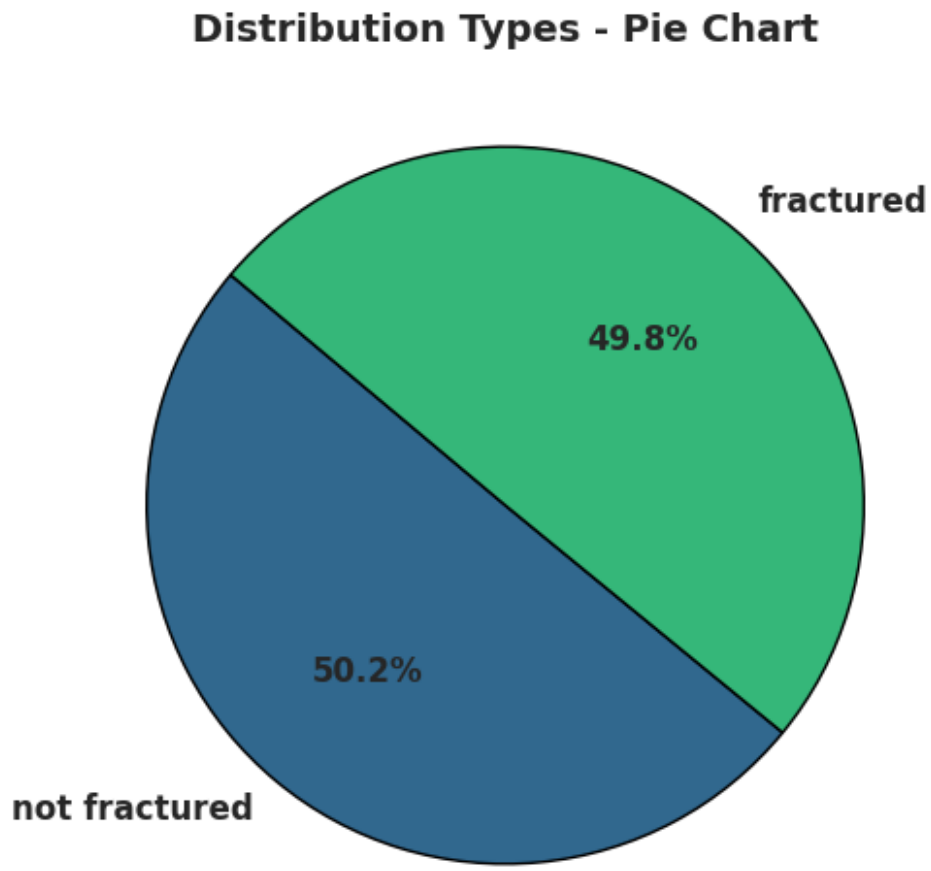
```
df['label'].value_counts()
```

```
label
not fractured    4640
fractured        4606
Name: count, dtype: int64
```

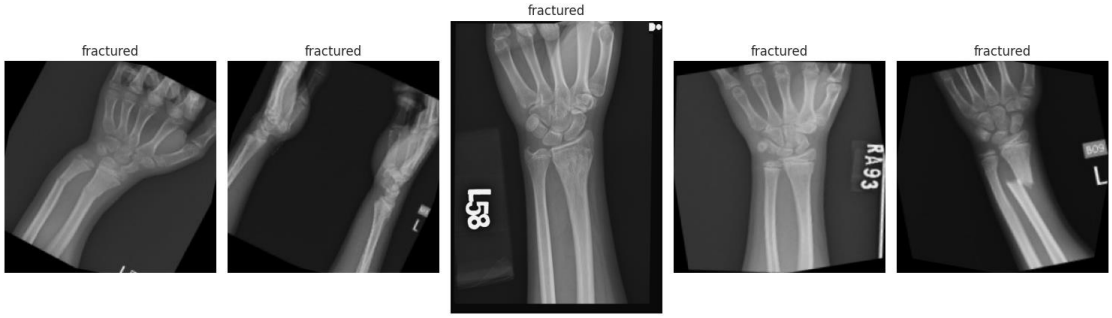
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.set_style("whitegrid")

fig, ax = plt.subplots(figsize=(8, 6))
sns.countplot(data=df, x="label", palette="viridis", ax=ax)

ax.set_title("Distribution Types", fontsize=14, fontweight='bold')
ax.set_xlabel("Tumor Type", fontsize=12)
ax.set_ylabel("Count", fontsize=12)
```

```
for p in ax.patches:
    ax.annotate(f'{int(p.get_height())}',
                (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='bottom', fontsize=11, color='black',
                xytext=(0, 5), textcoords='offset points')

plt.show()

label_counts = df["label"].value_counts()

fig, ax = plt.subplots(figsize=(8, 6))
colors = sns.color_palette("viridis", len(label_counts))

ax.pie(label_counts, labels=label_counts.index, autopct='%1.1f%%',
       startangle=140, colors=colors, textprops={'fontsize': 12, 'weight':
'bold'},
       wedgeprops={'edgecolor': 'black', 'linewidth': 1})

ax.set_title("Distribution Types - Pie Chart", fontsize=14,
fontweight='bold')

plt.show()
```






```
import cv2

num_images = 5

plt.figure(figsize=(15, 12))

for i, category in enumerate(categories):
    category_images = df[df['label'] ==
category]['image_path'].iloc[:num_images]

    for j, img_path in enumerate(category_images):

        img = cv2.imread(img_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        plt.subplot(len(categories), num_images, i * num_images + j + 1)
        plt.imshow(img)
        plt.axis('off')
        plt.title(category)

plt.tight_layout()
plt.show()
```



df

	image_path	label
0	/kaggle/input/fracture-multi-region-x-ray-data...	fractured
1	/kaggle/input/fracture-multi-region-x-ray-data...	fractured
2	/kaggle/input/fracture-multi-region-x-ray-data...	fractured
3	/kaggle/input/fracture-multi-region-x-ray-data...	fractured
4	/kaggle/input/fracture-multi-region-x-ray-data...	fractured
...	...	...
9241	/kaggle/input/fracture-multi-region-x-ray-data...	not fractured
9242	/kaggle/input/fracture-multi-region-x-ray-data...	not fractured
9243	/kaggle/input/fracture-multi-region-x-ray-data...	not fractured
9244	/kaggle/input/fracture-multi-region-x-ray-data...	not fractured
9245	/kaggle/input/fracture-multi-region-x-ray-data...	not fractured

[9246 rows x 2 columns]

```
import torch
import torch.nn as nn
import torchvision
import pandas as pd
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms
from PIL import Image, ImageFile
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns

ImageFile.LOAD_TRUNCATED_IMAGES = True
```

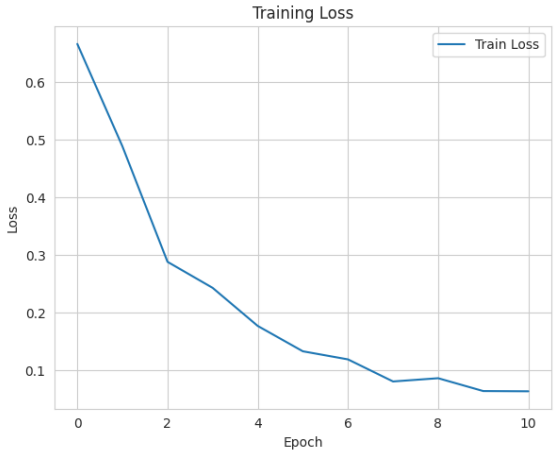
```
train_df, val_df = train_test_split(df, test_size=0.2, stratify=df['label'],
random_state=42)

class FractureDataset(Dataset):
    def __init__(self, df, transform=None):
        self.df = df
        self.transform = transform
        self.label_map = {'fractured': 1, 'not fractured': 0}
    def __len__(self): return len(self.df)
    def __getitem__(self, idx):
        row = self.df.iloc[idx]
        img = Image.open(row['image_path']).convert('RGB')
        label = self.label_map[row['label']]
        if self.transform: img = self.transform(img)
        return img, label

transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,
0.225])
])

train_dataset = FractureDataset(train_df, transform)
val_dataset   = FractureDataset(val_df,   transform)
train_loader  = DataLoader(train_dataset, batch_size=64, shuffle=True)
val_loader    = DataLoader(val_dataset,   batch_size=64)

model = torchvision.models.resnet18(weights='DEFAULT')
model.fc = nn.Linear(model.fc.in_features, 2)
model = model.cuda()

def synflow_pruning(model, sparsity=0.95, iters=100):
    model.train()
    for p in model.parameters():
        if p.dim() > 1: p.data = torch.abs(p.data)
    dummy = torch.randn(1, 3, 224, 224).cuda()
    for _ in range(iters):
        out = model(dummy)
        loss = out.sum()
        model.zero_grad()
        loss.backward()
        for p in model.parameters():
            if p.grad is not None and p.dim() > 1:
                p.data = p.data * torch.abs(p.grad)
    scores = {}
    for n, p in model.named_parameters():
        if p.dim() > 1: scores[n] = p.data.abs().flatten()
    total = sum(s.numel() for s in scores.values())
    thr = torch.topk(torch.cat([s for s in scores.values()]),
int(total*sparsity), largest=False)[0].max()
    for n, p in model.named_parameters():
```

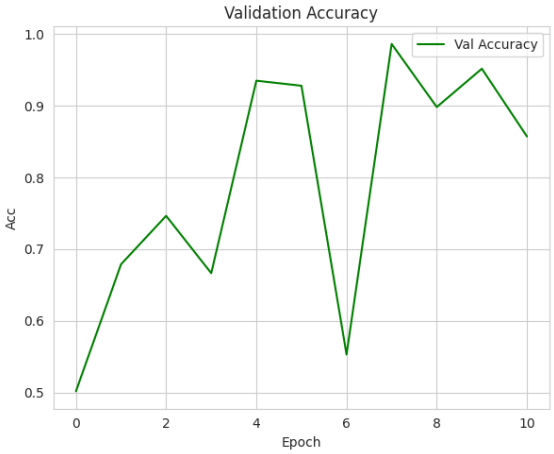
```
        if p.dim() > 1:
            mask = (p.data.abs() >= thr).float()
            p.data *= mask
    return model

model = synflow_pruning(model, sparsity=0.95)

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

train_losses = []
val_accs     = []
best_acc = 0.0
patience = 3
wait = 0

for epoch in range(30):
    model.train()
    epoch_loss = 0.0
    for X, y in train_loader:
        X, y = X.cuda(), y.cuda()
        optimizer.zero_grad()
        pred = model(X)
        loss = criterion(pred, y)
        loss.backward()
        optimizer.step()
        epoch_loss += loss.item()
    train_losses.append(epoch_loss/len(train_loader))

    model.eval()
    correct = 0
    with torch.no_grad():
        for X, y in val_loader:
            pred = model(X.cuda()).argmax(1)
            correct += (pred == y.cuda()).sum().item()
    acc = correct / len(val_dataset)
    val_accs.append(acc)
    print(f"Epoch {epoch+1:02d} | Loss: {train_losses[-1]:.4f} | Val Acc:
{acc:.4f}")

    if acc > best_acc:
        best_acc = acc
        torch.save(model.state_dict(), 'best_fracture_model.pth')
        wait = 0
    else:
        wait += 1
        if wait >= patience:
            print(f"Early stopping triggered at epoch {epoch+1}")
            break

model.load_state_dict(torch.load('best_fracture_model.pth'))
```

```
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot(train_losses, label='Train Loss')
plt.xlabel('Epoch'); plt.ylabel('Loss'); plt.legend(); plt.title('Training
Loss')
plt.subplot(1,2,2)
plt.plot(val_accs, label='Val Accuracy', color='green')
plt.xlabel('Epoch'); plt.ylabel('Acc'); plt.legend(); plt.title('Validation
Accuracy')
plt.tight_layout()
plt.savefig('curves.png')
plt.show()

all_preds, all_labels = [], []
model.eval()
with torch.no_grad():
    for X, y in val_loader:
        pred = model(X.cuda()).argmax(1).cpu().numpy()
        all_preds.extend(pred)
        all_labels.extend(y.numpy())

cm = confusion_matrix(all_labels, all_preds)
print("\n=== Classification Report ===")
print(classification_report(all_labels, all_preds, target_names=['not
fractured', 'fractured']))

plt.figure(figsize=(7,6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['not fractured','fractured'],
            yticklabels=['not fractured','fractured'])
plt.xlabel('Predicted'); plt.ylabel('True'); plt.title('Confusion Matrix')
plt.savefig('confusion_matrix.png')
plt.show()

final_acc = np.mean(np.array(all_preds) == np.array(all_labels))
print(f"\nFinal Val Accuracy: {final_acc:.4f}")
total_params = sum(p.numel() for p in model.parameters() if p.requires_grad)
print(f"Remaining parameters: {total_params:,}
({total_params/11_689_512*100:.1f}% of original)")

torch.save(model.state_dict(), 'fracture_detector_final.pth')
```

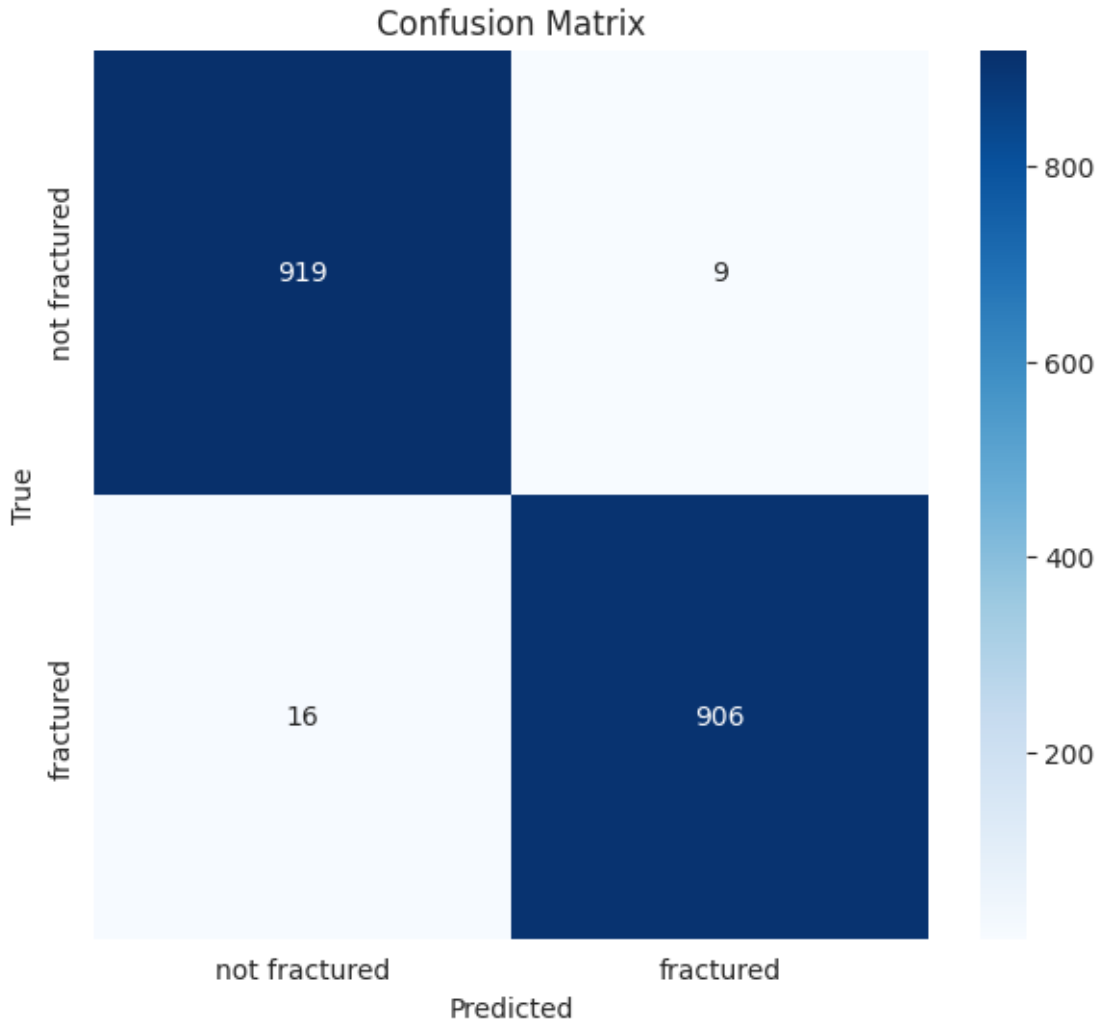
```
Epoch 01 | Loss: 0.6666 | Val Acc: 0.5016
Epoch 02 | Loss: 0.4888 | Val Acc: 0.6789
Epoch 03 | Loss: 0.2885 | Val Acc: 0.7465
Epoch 04 | Loss: 0.2433 | Val Acc: 0.6665
Epoch 05 | Loss: 0.1773 | Val Acc: 0.9351
Epoch 06 | Loss: 0.1333 | Val Acc: 0.9281
Epoch 07 | Loss: 0.1192 | Val Acc: 0.5530
Epoch 08 | Loss: 0.0809 | Val Acc: 0.9865
Epoch 09 | Loss: 0.0865 | Val Acc: 0.8984
```

```
Epoch 10 | Loss: 0.0643 | Val Acc: 0.9519
Epoch 11 | Loss: 0.0639 | Val Acc: 0.8573
Early stopping triggered at epoch 11
```






```
=== Classification Report ===
              precision    recall  f1-score   support

not fractured       0.98      0.99      0.99       928
    fractured       0.99      0.98      0.99       922

     accuracy                           0.99      1850
    macro avg       0.99      0.99      0.99      1850
 weighted avg       0.99      0.99      0.99      1850
```




```
Final Val Accuracy: 0.9865
Remaining parameters: 11,177,538 (95.6% of original)
```