S5 CSE Beta

EBY J KAVUNGAL

JOYAL GEORGE JOSEPH

JOES JACOB PAUL

```
from google.colab import files
files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving kaggle.json to kaggle.json
{'kaggle.json': b'{"username":"joyal22","key":"99fd809a3f10b091c5b63e176f7e17df"}'}
```

```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

```
cp: cannot stat 'kaggle.json': No such file or directory
chmod: cannot access '/root/.kaggle/kaggle.json': No such file or directory
```

```
!kaggle datasets download -d emmarex/plantdisease
```

```
Dataset URL: https://www.kaggle.com/datasets/emmarex/plantdisease
License(s): unknown
Downloading plantdisease.zip to /content
 99% 651M/658M [00:06<00:00, 124MB/s]
100% 658M/658M [00:06<00:00, 104MB/s]
```

```
!unzip plantdisease.zip -d plant_disease
```

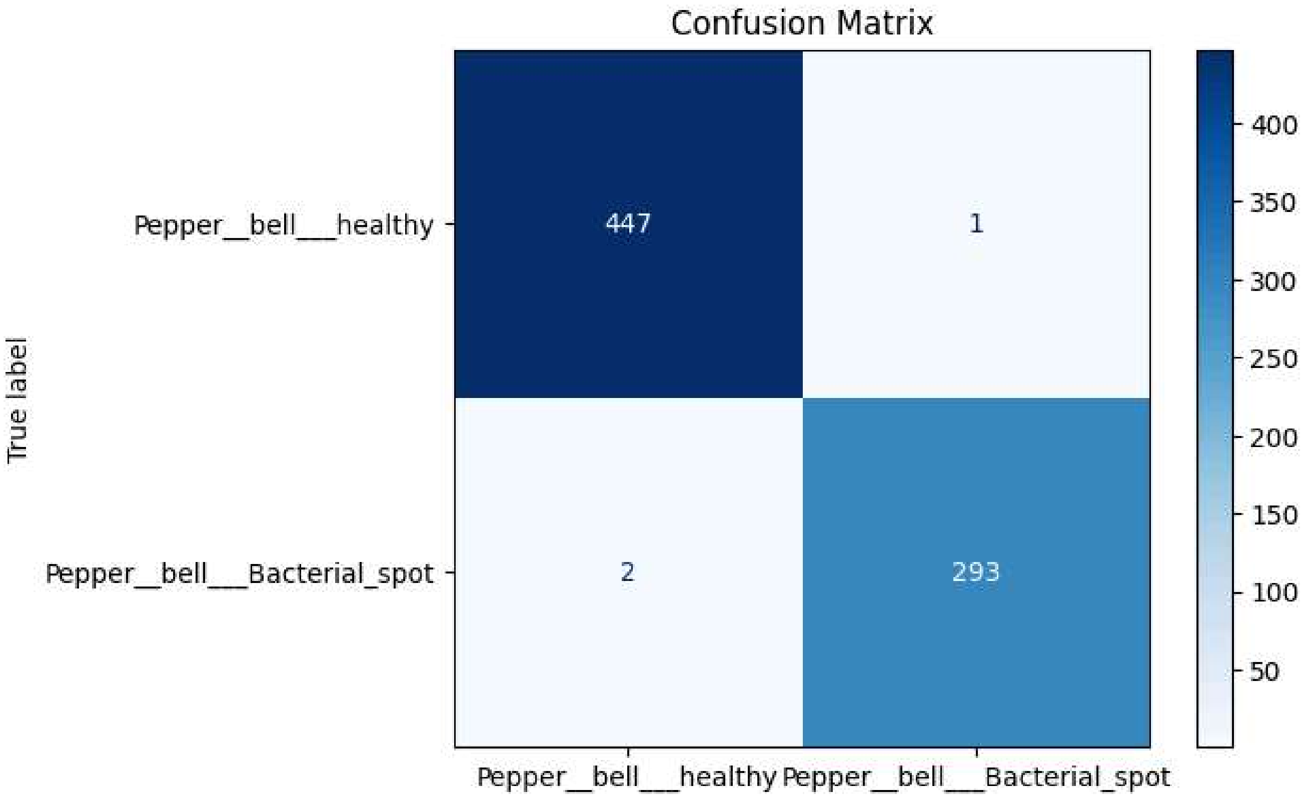
```
  inflating: plant_disease/plantvillage/PlantVillage/Tomato_healthy/fe28e4c7-0c35-4f52-984e-0e60f33a2c6e___GH_HL Leaf 198.JPG
  inflating: plant_disease/plantvillage/PlantVillage/Tomato_healthy/fe8f8808-2631-491e-a46b-bd2a1a4958e7___GH_HL Leaf 213.1.JPG
  inflating: plant_disease/plantvillage/PlantVillage/Tomato_healthy/feda8fd2-1d18-443e-a7d9-15bd6bf8ce66___RS_HL 0332.JPG
  inflating: plant_disease/plantvillage/PlantVillage/Tomato_healthy/ff354b62-5981-43d1-8cfe-ac58bc20ca20___GH_HL Leaf 221.JPG
  inflating: plant_disease/plantvillage/PlantVillage/Tomato_healthy/ff774aec-2504-4d11-8a61-2fd74c689a6f___RS_HL 9904.JPG
  inflating: plant_disease/plantvillage/PlantVillage/Tomato_healthy/ff8b36d5-feaf-4d2d-8126-18670a312657___RS_HL 0229.JPG
  inflating: plant_disease/plantvillage/PlantVillage/Tomato_healthy/ffb39943-eabb-42cf-ad09-b17019e46d66___RS_HL 9871.JPG
  inflating: plant_disease/plantvillage/PlantVillage/Tomato_healthy/ffd8aa68-138f-4114-96c7-21eef72e1e13___RS_HL 9881.JPG
```

```python
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import models, transforms
from torch.utils.data import Dataset, DataLoader, random_split
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from PIL import Image
import os
import matplotlib.pyplot as plt

# Define transformations
train_transforms = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

val_transforms = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

# Define a function to filter valid image files and ignore .ipynb_checkpoints
def is_valid_file(path):
    valid_extensions = {'.jpg', '.jpeg', '.png', '.ppm', '.bmp', '.pgm', '.tif', '.tiff', '.webp'}
    return (os.path.splitext(path)[1].lower() in valid_extensions
            and '.ipynb_checkpoints' not in path)
```

```
# Custom dataset class
class CustomImageFolder(Dataset):
    def __init__(self, root_dir, transform=None, is_valid_file=None):
        self.root_dir = root_dir
        self.transform = transform
        self.is_valid_file = is_valid_file
        self.image_paths = []
        self.labels = []

        for label in os.listdir(root_dir):
            class_path = os.path.join(root_dir, label)
            if os.path.isdir(class_path):
                for file_name in os.listdir(class_path):
                    file_path = os.path.join(class_path, file_name)
                    if is_valid_file(file_path):
                        self.image_paths.append(file_path)
                        self.labels.append(label)

        self.class_to_idx = {cls: idx for idx, cls in enumerate(os.listdir(root_dir)) if os.path.isdir(os.path.join(root_dir, cls))}

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

    def __getitem__(self, idx):
        img_path = self.image_paths[idx]
        label = self.class_to_idx[self.labels[idx]]
        img = Image.open(img_path).convert('RGB')

        if self.transform:
            img = self.transform(img)

        return img, label

# Path to your dataset directory
data_dir = '/content/Data'

# Load dataset using the custom class
dataset = CustomImageFolder(data_dir, transform=train_transforms, is_valid_file=is_valid_file)
```

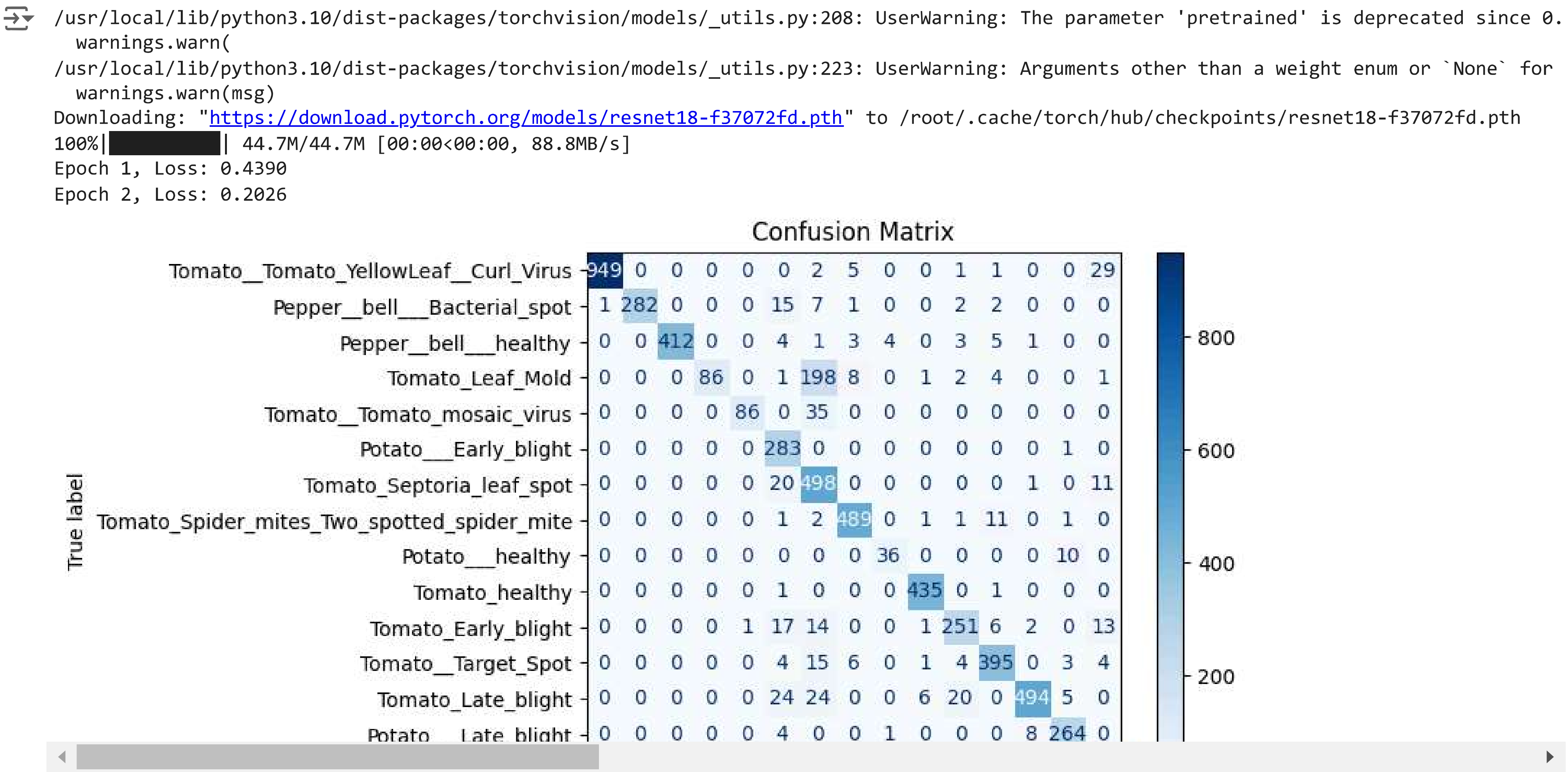
```
# Split dataset into training (70%) and testing (30%)
train_size = int(0.7 * len(dataset))
test_size = len(dataset) - train_size
train_dataset, test_dataset = random_split(dataset, [train_size, test_size])

# Apply validation transformations to the test dataset
test_dataset.dataset.transform = val_transforms

# Create data loaders
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)

# Load and modify ResNet-18 model
model = models.resnet18(pretrained=True)
num_features = model.fc.in_features
num_classes = len(dataset.class_to_idx)
model.fc = nn.Linear(num_features, num_classes)

# Move model to device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = model.to(device)

# Loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training loop
epochs =
for epoch in range(epochs):
    model.train()
    running_loss = 0.0

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

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

    print(f"Epoch {epoch+1}, Loss: {running_loss/len(train_loader):.4f}")

# Evaluate model and compute confusion matrix
model.eval()
all_labels = []
all_predictions = []

with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)
        all_labels.extend(labels.cpu().numpy())
        all_predictions.extend(predicted.cpu().numpy())

# Calculate confusion matrix
conf_matrix = confusion_matrix(all_labels, all_predictions)

# Plot confusion matrix
disp = ConfusionMatrixDisplay(conf_matrix, display_labels=list(dataset.class_to_idx.keys()))
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix")
plt.show()

# Save the trained model
torch.save(model.state_dict(), 'plant_disease_small.pth')

# Prediction function
def predict_image(image_path, model):
    model.eval()
    image = Image.open(image_path)
    transform = transforms.Compose([
        transforms.Resize((128, 128)),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])
    image = transform(image).unsqueeze(0).to(device)
    with torch.no_grad():
        output = model(image)
        _, predicted = torch.max(output, 1)
        class_idx = predicted.item()

    idx_to_class = {v: k for k, v in dataset.class_to_idx.items()}
    return idx_to_class[class_idx]
```

```
Epoch 1, Loss: 0.1431
Epoch 2, Loss: 0.0184
```



```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import models, transforms
from torch.utils.data import Dataset, DataLoader, random_split
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from PIL import Image
import os
import matplotlib.pyplot as plt

# Define transformations
train_transforms = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

val_transforms = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

# Define a function to filter valid image files and ignore .ipynb_checkpoints
def is_valid_file(path):
    valid_extensions = {'.jpg', '.jpeg', '.png', '.ppm', '.bmp', '.pgm', '.tif', '.
    return (os.path.splitext(path)[1].lower() in valid_extensions
            and '.ipynb_checkpoints' not in path)

# Custom dataset class
class CustomImageFolder(Dataset):
    def __init__(self, root_dir, transform=None, is_valid_file=None):
        self.root_dir = root_dir
        self.transform = transform
        self.is_valid_file = is_valid_file
        self.image_paths = []
        self.labels = []

        for label in os.listdir(root_dir):
            class_path = os.path.join(root_dir, label)
            if os.path.isdir(class_path):
                for file_name in os.listdir(class_path):
                    file_path = os.path.join(class_path, file_name)
                    if is_valid_file(file_path):
                        self.image_paths.append(file_path)
                        self.labels.append(label)

        self.class_to_idx = {cls: idx for idx, cls in enumerate(os.listdir(root_dir

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

    def __getitem__(self, idx):
        img_path = self.image_paths[idx]
        label = self.class_to_idx[self.labels[idx]]
        img = Image.open(img_path).convert('RGB')

        if self.transform:
            img = self.transform(img)

        return img, label

# Path to your dataset directory
data_dir = '/content/plant_disease/PlantVillage'

# Load dataset using the custom class
dataset = CustomImageFolder(data_dir, transform=train_transforms, is_valid_file=is_

# Split dataset into training (70%) and testing (30%)
train_size = int(0.7 * len(dataset))
test_size = len(dataset) - train_size
train_dataset, test_dataset = random_split(dataset, [train_size, test_size])

# Apply validation transformations to the test dataset
test_dataset.dataset.transform = val_transforms

# Create data loaders
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)

# Load and modify ResNet-18 model
model = models.resnet18(pretrained=True)
num_features = model.fc.in_features
```

```
num_classes = len(dataset.class_to_idx)
model.fc = nn.Linear(num_features, num_classes)

# Move model to device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = model.to(device)

# Loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training loop
epochs = 2
for epoch in range(epochs):
    model.train()
    running_loss = 0.0

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

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

        running_loss += loss.item()

    print(f"Epoch {epoch+1}, Loss: {running_loss/len(train_loader):.4f}")

# Evaluate model and compute confusion matrix
model.eval()
all_labels = []
all_predictions = []

with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)
        all_labels.extend(labels.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())
        all_predictions.extend(predicted.cpu().numpy())

# Calculate confusion matrix
conf_matrix = confusion_matrix(all_labels, all_predictions)

# Plot confusion matrix
disp = ConfusionMatrixDisplay(conf_matrix, display_labels=list(dataset.class_to_idx
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix")
plt.show()

# Save the trained model
torch.save(model.state_dict(), 'plant_disease.pth')

# Prediction function
def predict_image(image_path, model):
    model.eval()
    image = Image.open(image_path)
    transform = transforms.Compose([
        transforms.Resize((128, 128)),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])
    image = transform(image).unsqueeze(0).to(device)

    with torch.no_grad():
        output = model(image)
        _, predicted = torch.max(output, 1)
        class_idx = predicted.item()

    idx_to_class = {v: k for k, v in dataset.class_to_idx.items()}
    return idx_to_class[class_idx]
```



```
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since 0.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `None` for
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to /root/.cache/torch/hub/checkpoints/resnet18-f37072fd.pth
100%|██████████| 44.7M/44.7M [00:00<00:00, 88.8MB/s]
Epoch 1, Loss: 0.4390
Epoch 2, Loss: 0.2026
```

Confusion Matrix

Tomato__Tomato_YellowLeaf__Curl_Virus	949	0	0	0	0	0	2	5	0	0	1	1	0	0	29
Pepper__bell___Bacterial_spot	1	282	0	0	0	15	7	1	0	0	2	2	0	0	0
Pepper__bell___healthy	0	0	412	0	0	4	1	3	4	0	3	5	1	0	0
Tomato_Leaf_Mold	0	0	0	86	0	1	198	8	0	1	2	4	0	0	1
Tomato__Tomato_mosaic_virus	0	0	0	0	86	0	35	0	0	0	0	0	0	0	0
Potato___Early_blight	0	0	0	0	0	283	0	0	0	0	0	0	0	1	0
Tomato_Septoria_leaf_spot	0	0	0	0	0	20	498	0	0	0	0	0	1	0	11
Tomato_Spider_mites_Two_spotted_spider_mite	0	0	0	0	0	1	2	489	0	1	1	11	0	1	0
Potato___healthy	0	0	0	0	0	0	0	0	36	0	0	0	0	10	0
Tomato_healthy	0	0	0	0	0	1	0	0	0	435	0	1	0	0	0
Tomato_Early_blight	0	0	0	0	1	17	14	0	0	1	251	6	2	0	13
Tomato__Target_Spot	0	0	0	0	0	4	15	6	0	1	4	395	0	3	4
Tomato_Late_blight	0	0	0	0	0	24	24	0	0	6	20	0	494	5	0
Potato___Late_blight	0	0	0	0	0	4	0	0	1	0	0	0	8	264	0