In [15]:
```
from pathlib import Path

ROOT = Path("/Users/colinsohn/Downloads/lung_cancer")

adenos = list((ROOT / "adenocarcinoma").rglob("*.jpg"))
benigns = list((ROOT / "benign").rglob("*.jpg"))
squams = list((ROOT / "squamous_cell_carcinoma").rglob("*.jpg"))

print("adenocarcinoma:", len(adenos))
print("benign:", len(benigns))
print("squamous cell carcinoma:", len(squams))
print("total:", len(adenos) + len(benigns) + len(squams))
```

```
adenocarcinoma: 5000
benign: 5000
squamous cell carcinoma: 5000
total: 15000
```

In [19]:
```
from pathlib import Path
import random

import torch
from torch import nn, optim
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms

from PIL import Image
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score
import numpy as np
import matplotlib.pyplot as plt
```

In [20]:
```
# point to your dataset root
ROOT = Path("/Users/colinsohn/Downloads/lung_cancer")

class_names = ["adenocarcinoma", "benign", "squamous_cell_carcinoma"]
class_to_idx = {name: i for i, name in enumerate(class_names)}
class_to_idx
```

Out[20]:
```
{'adenocarcinoma': 0, 'benign': 1, 'squamous_cell_carcinoma': 2}
```

In [21]:
```
# build a list of (image_path, label_idx)
all_samples = []

for cls in class_names:
    folder = ROOT / cls
    files = list(folder.rglob("*.jpg"))
    label = class_to_idx[cls]
    for f in files:
        all_samples.append((f, label))

len(all_samples), all_samples[:5]
```

Out[21]:
```
(15000,
 [(PosixPath('/Users/colinsohn/Downloads/lung_cancer/adenocarcinoma/0071.jpg'),
   0),
  (PosixPath('/Users/colinsohn/Downloads/lung_cancer/adenocarcinoma/4217.jpg'),
   0),
  (PosixPath('/Users/colinsohn/Downloads/lung_cancer/adenocarcinoma/3578.jpg'),
   0),
  (PosixPath('/Users/colinsohn/Downloads/lung_cancer/adenocarcinoma/2666.jpg'),
   0),
  (PosixPath('/Users/colinsohn/Downloads/lung_cancer/adenocarcinoma/2100.jpg'),
   0)])
```

In [22]:
```
# shuffle for randomness
random.shuffle(all_samples)

train_val, test = train_test_split(all_samples, test_size=0.15, stratify=[y
train, val = train_test_split(train_val, test_size=0.1765, stratify=[y for _
# 0.1765 of 0.85 ≈ 0.15, so ~70/15/15 split

len(train), len(val), len(test)
```

Out[22]:
```
(10499, 2251, 2250)
```

In [23]:
```
IMAGE_SIZE = 224

train_tfms = transforms.Compose([
    transforms.Resize((IMAGE_SIZE, IMAGE_SIZE)),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    # simple normalization to [0,1], you can add mean/std later if you want
])

test_tfms = transforms.Compose([
    transforms.Resize((IMAGE_SIZE, IMAGE_SIZE)),
    transforms.ToTensor(),
])
```

In [24]:
```
class LungDataset(Dataset):
    def __init__(self, samples, transform=None):
        self.samples = samples
        self.transform = transform

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

    def __getitem__(self, idx):
        img_path, label = self.samples[idx]
        img = Image.open(img_path).convert("RGB")
        if self.transform is not None:
```

```
            img = self.transform(img)
        return img, label
```

In [25]:
```
train_ds = LungDataset(train, transform=train_tfms)
val_ds   = LungDataset(val,   transform=test_tfms)
test_ds  = LungDataset(test,  transform=test_tfms)

BATCH_SIZE = 32

train_loader = DataLoader(train_ds, batch_size=BATCH_SIZE, shuffle=True)
val_loader   = DataLoader(val_ds,   batch_size=BATCH_SIZE, shuffle=False)
test_loader  = DataLoader(test_ds,  batch_size=BATCH_SIZE, shuffle=False)
```

In [26]:
```
class SimpleCNN(nn.Module):
    def __init__(self, num_classes=3):
        super().__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 16, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),  # 224 -> 112

            nn.Conv2d(16, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),  # 112 -> 56

            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),  # 56 -> 28
        )

        # 64 feature maps of size 28x28
        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(64 * 28 * 28, 256),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(256, num_classes)
        )

    def forward(self, x):
        x = self.features(x)
        x = self.classifier(x)
        return x

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

Out[26]:
```
device(type='cpu')
```

In [27]:
```
model = SimpleCNN(num_classes=len(class_names)).to(device)

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

In [28]:
```python
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 = images.to(device)
        labels = labels.to(device)

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

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

    epoch_loss = running_loss / total
    epoch_acc = correct / total
    return epoch_loss, epoch_acc
```

In [29]:
```python
def evaluate(model, loader, criterion, device):
    model.eval()
    running_loss = 0.0
    correct = 0
    total = 0

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

            outputs = model(images)
            loss = criterion(outputs, labels)

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

    epoch_loss = running_loss / total
    epoch_acc = correct / total
    return epoch_loss, epoch_acc
```

In [30]:
```python
EPOCHS = 5  # try 5 first, then increase if needed

for epoch in range(1, EPOCHS + 1):
    train_loss, train_acc = train_one_epoch(model, train_loader, criterion,
    val_loss, val_acc = evaluate(model, val_loader, criterion, device)

    print(
```

```
        f"Epoch {epoch}/{EPOCHS} | "
        f"Train loss: {train_loss:.4f}, acc: {train_acc:.4f} | "
        f"Val loss: {val_loss:.4f}, acc: {val_acc:.4f}"
    )
```

```
Epoch 1/5 | Train loss: 0.5073, acc: 0.7410 | Val loss: 0.3590, acc: 0.8681
Epoch 2/5 | Train loss: 0.3434, acc: 0.8568 | Val loss: 0.2738, acc: 0.8907
Epoch 3/5 | Train loss: 0.2841, acc: 0.8837 | Val loss: 0.2604, acc: 0.8903
Epoch 4/5 | Train loss: 0.2574, acc: 0.8931 | Val loss: 0.2161, acc: 0.9116
Epoch 5/5 | Train loss: 0.2305, acc: 0.9088 | Val loss: 0.2353, acc: 0.8836
```

In [31]:
```
model.eval()
all_labels = []
all_preds = []

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

        outputs = model(images)
        preds = outputs.argmax(dim=1)

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

all_labels = np.array(all_labels)
all_preds = np.array(all_preds)

test_acc = accuracy_score(all_labels, all_preds)
cm = confusion_matrix(all_labels, all_preds)

print("Test accuracy:", test_acc)
print("Confusion matrix:\n", cm)
```

```
Test accuracy: 0.8884444444444445
Confusion matrix:
 [[570   1 179]
 [ 32 718   0]
 [ 39   0 711]]
```

In [34]:
```
def plot_confusion_matrix(cm, class_names):
    fig, ax = plt.subplots(figsize=(5, 5))

    # Pink/purple colormap
    im = ax.imshow(cm, interpolation="nearest", cmap="PuRd")
    ax.figure.colorbar(im, ax=ax)

    ax.set(
        xticks=np.arange(len(class_names)),
        yticks=np.arange(len(class_names)),
        xticklabels=class_names,
        yticklabels=class_names,
    )
    ax.set_xlabel("Predicted label")
    ax.set_ylabel("True label")
    ax.set_title("Confusion Matrix")
```

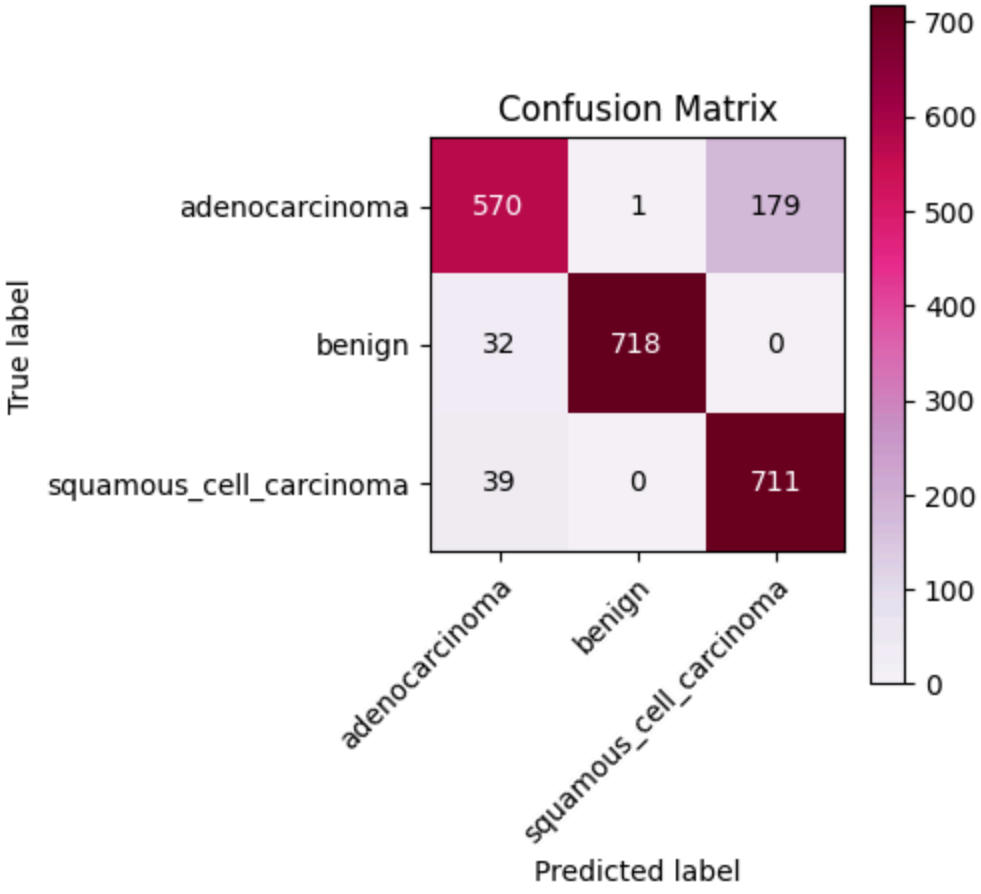
```
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right", rotation_mode="a

    # Adjust text color based on intensity
    max_val = cm.max()
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            value = cm[i, j]
            text_color = "white" if value > max_val * 0.5 else "black"
            ax.text(j, i, format(value, "d"),
                    ha="center", va="center",
                    color=text_color)

    fig.tight_layout()
    plt.show()



plot_confusion_matrix(cm, class_names)
```



In [ ]:
```
#
```