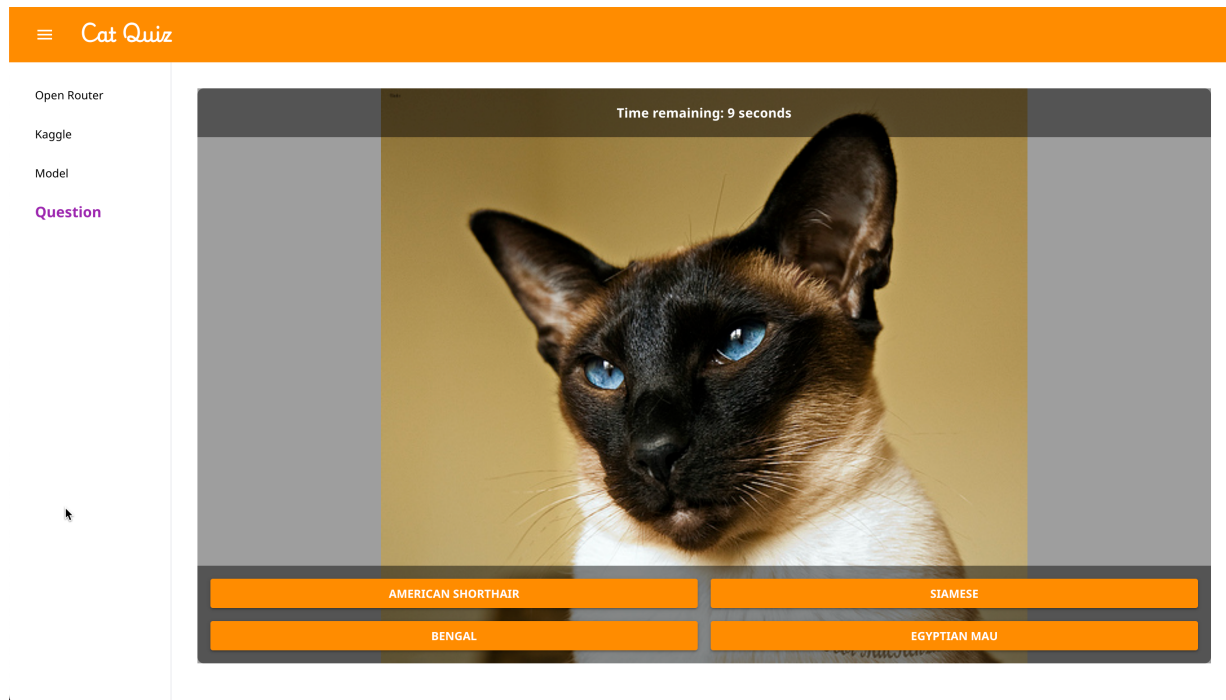
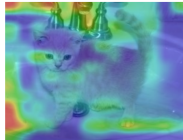
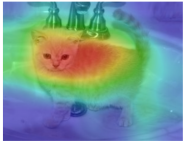
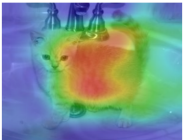
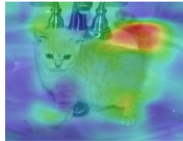


Cat Breed Classifier

This project is to train image classifier model to classify Cat Breeds dataset images. It has frontend as web application showing Quiz which allow user to take on guessing cat breed on the images.



Then after user pick answer, it will show result comparing user's with ChatGPT-4o-mini (OpenRouter API key required) and 4 Deep Learning Models. The rationale of LLM and GradCAM has been implemented to show model interpretability.

<p>Your Answer</p> <p>British Shorthair ✓ Correct!</p>	<p>ChatGPT-4o-mini Answer (1.53 seconds)</p> <p>British Shorthair ✓ Correct!</p> <p>The cat appears to have a stocky build and round face with short fur, which are characteristics typical of the British Shorthair breed.</p>	<p>Classification Model Answer (0.10 seconds)</p> <p>I don't know ✗ Incorrect!</p> <p>The correct answer was: British Shorthair</p> 
<p>ResNet Model Answer (0.11 seconds)</p> <p>Persian ✗ Incorrect!</p> <p>The correct answer was: British Shorthair</p> 	<p>EfficientNet Model Answer (0.43 seconds)</p> <p>British Shorthair ✓ Correct!</p> 	<p>VGG Model Answer (0.13 seconds)</p> <p>I don't know ✗ Incorrect!</p> <p>The correct answer was: British Shorthair</p> 

You : 2 ChatGPT : 2 Classification : 0 ResNet : 1 EfficientNet : 2 VGG : 1

[NEXT QUESTION](#)

Data Sets

- Dataset is from Kaggle from 'Geno Cat Breed Image Collection' dataset (url: <https://www.kaggle.com/datasets/shawngano/gano-cat-breed-image-collection>)
- Contains 15 cat breeds with 375 photos for each breed (total 5,625 photos)
- Preprocessing step
 - Resize to 256x256
 - Random Crop to 224x224
 - Random Horizontal Flip
 - Random Rotation
 - Convert to Tensor
 - Normalize with mean and std of ImageNet

Model Architecture

Custom Model (1 Model)

1. CNN Model

- 5 Convolutional Layers
- 2 Fully Connected Layers
- 1 Dropout Layer in between the two FC Layers

from src/pages.py

```
[17]: import os
import time
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Dataset
import torchvision
import torchvision.transforms as transforms

class CatBreedClassifier(nn.Module):
    def __init__(self):
        super(CatBreedClassifier, self).__init__()
        self.conv_block = torch.nn.Sequential(
            torch.nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2), # Output: 64 x 112 x 112
            torch.nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2), # Output: 128 x 56 x 56
            torch.nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2), # Output: 256 x 28 x 28
            torch.nn.Conv2d(256, 512, kernel_size=3, stride=1, padding=1),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2), # Output: 512 x 14 x 14
            torch.nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1), # Changed
            ↪ from 1024 to 512
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2) # Output: 512 x 7 x 7
```

```

    )

    # Calculate the flattened size: 512 channels * 7 * 7 = 25088
    self.fc_block = torch.nn.Sequential(
        torch.nn.Flatten(),
        torch.nn.Linear(512 * 7 * 7, 512), # Adjusted input size
        torch.nn.ReLU(),
        torch.nn.Dropout(0.5),
        torch.nn.Linear(512, 15), # 15 breeds
        torch.nn.LogSoftmax(dim=1)
    )

    def forward(self, x):
        x = self.conv_block(x)
        x = self.fc_block(x)
        return x

```

Transfer Learning (3 models)

2. VGG 16 Model

```

[18]: def get_vgg_model() -> nn.Module:
        vgg_model = torchvision.models.vgg16(weights=torchvision.models.VGG16_Weights.
↳ IMAGENET1K_V1)
        for param in vgg_model.parameters():
            param.requires_grad = False
        vgg_model.classifier[6] = nn.Linear(vgg_model.classifier[6].in_features, 15)
        return vgg_model

```

3. ResNet 18 Model

```

[19]: def get_resnet_model() -> nn.Module:
        res_model = torchvision.models.resnet18(weights=torchvision.models.
↳ ResNet18_Weights.IMAGENET1K_V1)
        for param in res_model.parameters():
            param.requires_grad = False
        res_model.fc = nn.Linear(res_model.fc.in_features, 15)
        return res_model

```

4. EfficientNet B2 Model

```

[20]: def get_efficient_net_model() -> nn.Module:
        eff_model = torchvision.models.efficientnet_b2(weights=torchvision.models.
↳ EfficientNet_B2_Weights.IMAGENET1K_V1)
        for param in eff_model.parameters():
            param.requires_grad = False
        eff_model.classifier[1] = nn.Linear(eff_model.classifier[1].in_features, 15)
        return eff_model

```

Training

All 4 models has been trained using same hyperparamters for comparison.

- Training Batch Size: 128
- Learning Rate: 0.001
- Momentum: 0.9
- Epochs: 5
- Loss Function: Cross Entropy Loss
- Optimizer: SGD

After finish training model saved to .pth for later used for inference result on Cat Quiz program.

 classification_model.pth	65,514 KB	PTH File
 efficientnet_model.pth	30,830 KB	PTH File
 resnet_model.pth	43,818 KB	PTH File
 vgg_model.pth	524,734 KB	PTH File

Below code is from src\model_train.py use in training all 4 models.

- CatBreedDataSet = Custom Dataset for Kaggle Cat Breed dataset
- get_cat_breed_dataset = function to create new catbreed dataset for training.

```
[21]: class CatBreedDataset(Dataset):
    def __init__(self, image_path, class_names, transform):
        self.image_path = image_path
        self.class_names = class_names
        self.class_to_idx = {class_name: i for i, class_name in enumerate(class_names)}
        self.transform = transform
        self.image_files = []
        self.image_labels = []
        self.image_tensor = []
        for class_name in self.class_names:
            for image_file in os.listdir(os.path.join(self.image_path, class_name)):
                self.image_files.append(os.path.join(self.image_path, class_name,
↵image_file))
                self.image_labels.append(self.class_to_idx[class_name])
                self.image_tensor.append(self.transform(Image.open(os.path.join(self.
↵image_path, class_name, image_file)).convert('RGB'))))

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

    def __getitem__(self, idx):
        image_tensor = self.image_tensor[idx]
        image_label = self.image_labels[idx]
        return image_tensor, image_label

def get_cat_breed_dataset(app_state, transform) -> CatBreedDataset:
    return CatBreedDataset(app_state.image_path, app_state.class_names, transform)
```

- get_transformation = function to build transformation using imagenet normalization.

```
[22]: def get_transformation() -> transforms.Compose:
    return transforms.Compose(
        [transforms.Resize((256, 256)),
         transforms.RandomCrop((224, 224)),
         transforms.RandomHorizontalFlip(),
         transforms.RandomRotation(10),
         transforms.ToTensor(),
         transforms.Normalize(mean=[0.485, 0.456, 0.406],
                              std=[0.229, 0.224, 0.225])]
    )
```

- train_model = function to train model and saved training accuracy/loss and validation accuracy/loss for later display.

```
[23]: def train_model(app_state,model: nn.Module, model_name: str) -> tuple[list[float],
↳list[float], list[float], float]:
    print(f"Training model on {app_state.device}")
    transform = get_transformation()

    print(f"Loading dataset")
    dataset = get_cat_breed_dataset(app_state, transform)

    # Split dataset into train and test sets (80% train, 20% test)
    train_size = int(0.8 * len(dataset))
    test_size = len(dataset) - train_size
    train_dataset, test_dataset = torch.utils.data.random_split(dataset, [train_size,
↳test_size])

    # Create data loaders for both sets
    train_loader = DataLoader(train_dataset, batch_size=128, shuffle=True,
↳num_workers=4)
    test_loader = DataLoader(test_dataset, batch_size=128, shuffle=False,
↳num_workers=4)

    print(f"Dataset split: {train_size} training samples, {test_size} test samples")

    print(f"Loading model")
    model.to(app_state.device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)

    training_loss = []
    training_accuracy = []
    validation_loss = []
    validation_accuracy = []

    start_time = time.time()
    for epoch in range(5):
        model.train()

        running_loss = 0.0
```

```

correct = 0
total = 0
for i, data in enumerate(train_loader, 0):
    inputs, labels = data
    inputs = inputs.to(app_state.device)
    labels = labels.to(app_state.device)
    optimizer.zero_grad()

    outputs = model(inputs)
    loss = criterion(outputs, labels)
    loss.backward()
    optimizer.step()
    running_loss += loss.item()
    _, predicted = torch.max(outputs.data, 1)
    correct += (predicted == labels).sum().item()
    total += labels.size(0)

    print(f"Epoch {epoch + 1}, Batch {i + 1}, Loss: {loss.item()}")

training_loss.append(running_loss / len(train_loader))
training_accuracy.append(correct / total)

model.eval()
with torch.no_grad():
    running_loss = 0.0
    correct = 0
    total = 0
    for i, data in enumerate(test_loader, 0):
        inputs, labels = data
        inputs = inputs.to(app_state.device)
        labels = labels.to(app_state.device)
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        running_loss += loss.item()
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    validation_loss.append(running_loss / len(test_loader))
    validation_accuracy.append(correct / total)

    print(f"Accuracy of the network on the {total} test images: {100 * correct /
total}%")
print("Finished Training")
print(f"Loss: {running_loss / len(train_loader)}")
training_time = time.time() - start_time
print(f"Saving model to {model_name}")
scripted_model = torch.jit.script(model)
torch.jit.save(scripted_model, model_name)

```

```
    return training_loss, training_accuracy, validation_loss, validation_accuracy, \
           training_time
```

Result and Evaluation

```
[24]: ! jupyter nbconvert --to pdf Wiwat_Pytorch_1254311.ipynb --template latex_authentic
```

```
[NbConvertApp] Converting notebook Wiwat_Pytorch_1254311.ipynb to pdf
[NbConvertApp] Writing 52726 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | b had problems, most likely because there were no citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 64663 bytes to Wiwat_Pytorch_1254311.pdf
```