Line-by-Line Explanation of Python Modules for Multimodal Breast Cancer Detection

Included Python Files

1. **`transformer_model.py`: A Transformer-based model for analyzing clinical text data (e.g., patient history and clinical notes).**
2. **`multimodal_model.py`: A combined CNN + Transformer model for integrating imaging and textual data for improved breast cancer detection.**
3. **`cnn_model.py`: A Convolutional Neural Network (CNN) for processing breast cancer imaging data such as mammograms, ultrasounds, and MRI scans.**

`transformer_model.py`

---

1. Imports

python code

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torchvision.models import resnet50
from transformers import AutoModel, AutoTokenizer
```

- **torch**, **torch.nn**, **torch.nn.functional**:
  PyTorch library for defining and training neural networks.
- **resnet50**:
  Imports the ResNet-50 architecture from `torchvision.models` to serve as an image feature extractor.

- **AutoModel, AutoTokenizer**:
  From Hugging Face's Transformers library, used to load pre-trained transformer models (e.g., BERT) and their corresponding tokenizers for text data.

---

2. Class Definition

python code

```
class MultimodalTransformer(nn.Module):
    def __init__(self, num_classes=2,
text_model_name="bert-base-uncased", hidden_dim=512):
        super(MultimodalTransformer, self).__init__()
```

- **MultimodalTransformer**:
  Defines the Transformer-based model for fusing image and text features.
- **`__init__()`**:
  Initializes the model's layers.
- **Parameters**:
  - `num_classes`: Number of output classes (e.g., cancerous or non-cancerous).
  - `text_model_name`: Pre-trained text model name from Hugging Face (e.g., "bert-base-uncased").
  - `hidden_dim`: Size of the hidden representation used in the Transformer.

---

3. Image Feature Extractor

python code

```
self.image_encoder = resnet50(pretrained=True)
self.image_encoder.fc = nn.Identity()  # Remove the classification
head
```

- **`self.image_encoder`**:
  ResNet-50 model is used to extract features from images.
- **`pretrained=True`**:
  Loads ResNet-50 pre-trained on ImageNet for faster convergence.
- **`self.image_encoder.fc = nn.Identity()`**:
  Replaces the final fully connected (classification) layer with an identity layer, so the model outputs raw features (size: `(B, 2048)`).

---

4. Text Feature Extractor

python code

```
self.text_encoder = AutoModel.from_pretrained(text_model_name)
self.text_tokenizer = AutoTokenizer.from_pretrained(text_model_name)
self.text_fc = nn.Linear(self.text_encoder.config.hidden_size,
hidden_dim)
```

- **`self.text_encoder`**:
  Loads a pre-trained transformer (e.g., BERT) to process clinical notes or patient history.
- **`self.text_tokenizer`**:
  Tokenizer that converts text into input IDs, attention masks, etc., required by the transformer model.
- **`self.text_fc`**:
  A fully connected layer that maps the transformer's output (size: `768` for BERT) to the same dimensionality as the image features (`hidden_dim`).

---

5. Multimodal Transformer Encoder

python code

```
self.transformer = nn.Transformer(
    d_model=hidden_dim,
    nhead=8,
    num_encoder_layers=6,
    num_decoder_layers=6,
    dim_feedforward=1024,
    dropout=0.1
)
```

- **`self.transformer`**:
  A standard PyTorch Transformer that fuses image and text features into a unified representation.
- **Key Parameters**:
  - `d_model=hidden_dim`: Embedding size of input features for the transformer.
  - `nhead=8`: Number of attention heads in the multi-head attention mechanism.
  - `num_encoder_layers=6`: Number of transformer encoder layers.
  - `dim_feedforward=1024`: Size of the feedforward layers inside the transformer.
  - `dropout=0.1`: Dropout rate for regularization.

6. Classification Head

python code

```
self.classifier = nn.Sequential(
    nn.Linear(hidden_dim, 256),
    nn.ReLU(),
    nn.Dropout(0.3),
    nn.Linear(256, num_classes)
)
```

- **`self.classifier`**:
  A feedforward network that maps the output of the transformer to the final class logits.
- **Components**:
  - `Linear(hidden_dim, 256)`: Reduces the transformer output to 256 features.
  - `ReLU()`: Applies non-linear activation.
  - `Dropout(0.3)`: Prevents overfitting by randomly deactivating 30% of neurons.
  - `Linear(256, num_classes)`: Produces logits for each class.

---

7. Forward Method

python code

```
def forward(self, image, text):
```

- **Purpose**: Defines how data flows through the model during a forward pass.

Image Features

python code

```
image_features = self.image_encoder(image)  # (B, 2048)
image_features = image_features.unsqueeze(1)  # Add sequence dimension
-> (B, 1, 2048)
```

- Extracts features from images using the ResNet-50 encoder.
- Adds a sequence dimension (`unsqueeze(1)`) to make the features compatible with the transformer input format.

Text Features

python code

```
text_inputs = self.text_tokenizer(text, return_tensors="pt",
padding=True, truncation=True, max_length=512)
text_outputs = self.text_encoder(**{k: v.to(image.device) for k, v in
text_inputs.items()})
text_features = self.text_fc(text_outputs.last_hidden_state[:, 0, :])
# CLS token -> (B, hidden_dim)
```

- **Tokenization**:
  Converts text into token IDs, attention masks, and other inputs required by the transformer.
- **`last_hidden_state[:, 0, :]`**:
  Extracts the `[CLS]` token's output, which contains the text's aggregated representation.
- **`self.text_fc`**:
  Maps the `[CLS]` representation to the same dimensionality as the image features.

Multimodal Fusion

python code

```
multimodal_input = torch.cat([image_features,
text_features.unsqueeze(1)], dim=1)  # (B, 2, hidden_dim)
multimodal_output = self.transformer(multimodal_input,
multimodal_input)  # (B, 2, hidden_dim)
```

- **Concatenation**:
  Combines image features and text features into a sequence (length: 2).
  Shape: `(B, 2, hidden_dim)`.
- **Transformer Encoder**:
  Processes the combined input to create fused multimodal representations.

Classification

python code

```
logits = self.classifier(multimodal_output[:, 0, :])  # (B,
num_classes)
```

- **`multimodal_output[:, 0, :]`**:
  Selects the first token's output (image token) for classification.
- **`self.classifier`**:
  Produces logits for each class (e.g., cancerous or non-cancerous).

---

8. Example Usage

python code

```
if __name__ == "__main__":
    model = MultimodalTransformer(num_classes=2)
    model.eval()
    sample_image = torch.randn(2, 3, 224, 224)  # Batch of 2 RGB
images
    sample_text = ["Patient history suggests a high risk of breast
cancer.",
                   "Clinical notes indicate benign tumor
characteristics."]
    with torch.no_grad():
        output = model(sample_image, sample_text)
        print("Logits:", output)
```

- **Initialize the Model**: Creates an instance of `MultimodalTransformer`.
- **Input Data**:
  - `sample_image`: A batch of random images (size `(B, C, H, W)`).
  - `sample_text`: Clinical notes corresponding to each image.
- **Inference**:
  Produces logits indicating the likelihood of each class.

multimodal_model.py

Import Libraries

python code

```
import torch
import torch.nn as nn
import torch.nn.functional as F
```

```python
from torchvision.models import resnet50
from transformers import AutoModel, AutoTokenizer
```

- `torch` and `torch.nn`: Used for defining and managing neural networks.
- `torch.nn.functional`: Contains functions for activation, loss, etc.
- `resnet50`: Pre-trained ResNet-50 CNN model from `torchvision` for image feature extraction.
- `AutoModel` and `AutoTokenizer`: Hugging Face tools to load a pre-trained Transformer model (e.g., BERT) and its tokenizer for text processing.

Model Definition

python code

```python
class MultimodalModel(nn.Module):
    def __init__(self, num_classes=2,
text_model_name="bert-base-uncased", hidden_dim=512):
```

- `MultimodalModel`: Class representing the multimodal model.
- `num_classes`: Number of classes for the classification task (default is 2, e.g., benign vs malignant).
- `text_model_name`: Name of the Hugging Face pre-trained transformer model (default is `bert-base-uncased`).
- `hidden_dim`: Dimensionality of hidden layers for text and multimodal fusion.

Image Encoder

python code

```python
self.image_encoder = resnet50(pretrained=True)
self.image_encoder.fc = nn.Identity()
```

- **`self.image_encoder`**: A ResNet-50 model pre-trained on ImageNet.
- **`self.image_encoder.fc`**: Replaces ResNet's classification head with an identity layer to output raw features instead of predictions.

Output: Image features of size `(batch_size, 2048)`.

Text Encoder

python code

```
self.text_encoder = AutoModel.from_pretrained(text_model_name)
self.text_tokenizer = AutoTokenizer.from_pretrained(text_model_name)
self.text_fc = nn.Linear(self.text_encoder.config.hidden_size,
hidden_dim)
```

- **`self.text_encoder`**: A pre-trained transformer model (e.g., BERT) from Hugging Face.
- **`self.text_tokenizer`**: Tokenizer to convert clinical notes into input IDs, attention masks, etc.
- **`self.text_fc`**: A fully connected layer to reduce the text feature size from the transformer's hidden size to `hidden_dim`.

Output: Text embeddings of size `(batch_size, hidden_dim)`.

Multimodal Fusion Transformer

python code

```
self.fusion_transformer = nn.Transformer(
    d_model=hidden_dim,
    nhead=8,
    num_encoder_layers=4,
    num_decoder_layers=4,
    dim_feedforward=1024,
    dropout=0.1
)
```

- **`nn.Transformer`**: A PyTorch Transformer model for fusing image and text features.
- **`d_model`**: Input feature size (set to `hidden_dim`).
- **`nhead`**: Number of attention heads (8 heads).
- **`num_encoder_layers`/`num_decoder_layers`**: Number of encoder and decoder layers (4 each).
- **`dim_feedforward`**: Dimensionality of the feedforward sub-layer in each Transformer block (1024).
- **`dropout`**: Dropout rate for regularization (0.1).

Output: Fused representation of the image and text features.

Classification Head

python code

```
self.classifier = nn.Sequential(
    nn.Linear(hidden_dim, 256),
    nn.ReLU(),
    nn.Dropout(0.3),
    nn.Linear(256, num_classes)
)
```

- **`nn.Sequential`**: A stack of layers for the classification task:
  - Fully connected layer: Reduces `hidden_dim` to 256.
  - ReLU activation: Introduces non-linearity.
  - Dropout (0.3): Regularization to prevent overfitting.
  - Final fully connected layer: Outputs `num_classes` logits (e.g., malignant vs benign).

Forward Pass

python code

```
def forward(self, images, texts):
```

- **`forward`**: Defines how the model processes input data.

Step 1: Process Images

python code

```
image_features = self.image_encoder(images)
image_features = image_features.unsqueeze(1)
```

- Passes input images through the ResNet-50 model to extract features `(batch_size, 2048)`.
- Adds a sequence dimension using `unsqueeze(1)`, resulting in shape `(batch_size, 1, 2048)`.

Step 2: Process Text

python code

```python
text_inputs = self.text_tokenizer(texts, return_tensors="pt",
padding=True, truncation=True, max_length=512)
text_inputs = {k: v.to(images.device) for k, v in text_inputs.items()}
text_outputs = self.text_encoder(**text_inputs)
text_features = self.text_fc(text_outputs.last_hidden_state[:, 0, :])
```

- Tokenizes clinical notes into input IDs, attention masks, etc.
- Moves tokenized data to the same device (CPU/GPU) as the images.
- Extracts hidden states from the transformer and takes the `[CLS]` token embedding (`last_hidden_state[:, 0, :]`).
- Reduces text feature size to `hidden_dim` using `self.text_fc`.

Output: Text features of size `(batch_size, hidden_dim)`.

Step 3: Fuse Image and Text Features
python code
```python
multimodal_input = torch.cat([image_features,
text_features.unsqueeze(1)], dim=1)
fused_output = self.fusion_transformer(multimodal_input,
multimodal_input)
```

- Concatenates image and text features along the sequence dimension, resulting in shape `(batch_size, 2, hidden_dim)`.
- Passes the combined sequence through the Transformer for multimodal fusion.

Output: Fused features of shape `(batch_size, 2, hidden_dim)`.

Step 4: Classification
python code
```python
logits = self.classifier(fused_output[:, 0, :])
```

- Uses the first token (image features) from the fused output for classification.
- Passes this token through the classifier to output logits `(batch_size, num_classes)`.

Testing the Model

python code

```
if __name__ == "__main__":
    model = MultimodalModel(num_classes=2)
    model.eval()
    sample_images = torch.randn(2, 3, 224, 224)
    sample_texts = [
        "The patient has a high probability of malignancy based on
prior biopsy results.",
        "Benign tumor detected with no signs of aggressive growth."
    ]
    with torch.no_grad():
        output = model(sample_images, sample_texts)
        print("Predicted logits:", output)
```

- Initializes the model and switches it to evaluation mode.
- Creates two sample inputs:
    - **Images**: Random tensors with shape `(2, 3, 224, 224)` (batch of 2 images).
    - **Texts**: Clinical notes as a list of strings.
- Performs inference and prints the predicted logits.

Summary of Key Components

1. **Image Encoder**: Extracts deep image features using ResNet-50.
2. **Text Encoder**: Encodes clinical notes with a Transformer (e.g., BERT).
3. **Multimodal Fusion**: Combines image and text features via a Transformer.
4. **Classification Head**: Outputs logits for binary or multi-class predictions.

CNN

---

1. Import Libraries

python code

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torchvision.models import resnet50
```

- **torch** **and** **torch.nn**: For creating and managing PyTorch neural networks.
- **torch.nn.functional**: Provides functions like activation, pooling, etc.
- **resnet50**: Pre-trained ResNet-50 architecture from torchvision.

---

2. Class Definition

python code

```
class CNNModel(nn.Module):
    def __init__(self, num_classes=2, pretrained=True):
```

- **CNNModel**: Defines the CNN-based model.
- **num_classes**: Number of classes for classification (e.g., benign vs malignant).
- **pretrained**: Whether to load pre-trained ResNet-50 weights (useful for transfer learning).

---

3. Backbone Network

python code

```
self.backbone = resnet50(pretrained=pretrained)
self.backbone.fc = nn.Identity()
```

- **self.backbone**: A ResNet-50 network pre-trained on ImageNet.
- **self.backbone.fc**: Replaces the fully connected (classification) head with an identity layer to output raw features.

Output: Feature embeddings of size (batch_size, 2048).

---

4. Classification Head

python code

```
self.classifier = nn.Sequential(
    nn.Linear(2048, 512),  # First layer: reduces 2048 features to 512
    nn.ReLU(),             # Non-linear activation
    nn.Dropout(0.3),       # Dropout for regularization
    nn.Linear(512, num_classes)  # Final layer: outputs logits for
each class
)
```

- `nn.Linear(2048, 512)`: Reduces the feature vector from ResNet to 512 dimensions.
- `nn.ReLU()`: Applies the ReLU activation function for non-linearity.
- `nn.Dropout(0.3)`: Adds dropout with a rate of 30% to prevent overfitting.
- `nn.Linear(512, num_classes)`: Outputs logits for classification.

---

5. Forward Pass

python code

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

- **Input (`x`)**: A batch of images of size `(batch_size, 3, H, W)` (e.g., 224x224 RGB images).
- `self.backbone(x)`: Extracts image features using ResNet.
- `self.classifier(features)`: Passes the extracted features through the classification head.
- **Output (`logits`)**: Predicted logits for each class, shape `(batch_size, num_classes)`.

---

6. Testing the Model

python code

```
if __name__ == "__main__":
    model = CNNModel(num_classes=2, pretrained=False)
    model.eval()

    sample_images = torch.randn(2, 3, 224, 224)

    with torch.no_grad():
        output = model(sample_images)
        print("Predicted logits:", output)
```

- `model = CNNModel(num_classes=2)`: Initializes the CNN model for binary classification.

- **`model.eval()`**: Switches the model to evaluation mode (disables dropout, etc.).
- **`sample_images`**: Generates a batch of random input images of size `(2, 3, 224, 224)`.
- **`output`**: Predicted logits for the two input images.

Expected Output

When run, the script will output something like:

lua
```
Predicted logits: tensor([[ 0.1243, -0.1876],
                         [ 0.2514, -0.0967]])
```

Each row corresponds to the unnormalized logits (scores) for each class.

Use Case

This model is suitable for:

- Image classification tasks (e.g., benign vs malignant).
- Serving as the image encoder in a multimodal pipeline (e.g., combined with clinical text data).