INFOKUM
Volume 13, Number 05, 2025, DOI 10.58471/infokum.v13i05
ESSN 2722-4635 (Online)
https://infor.seaninstitute.org/index.php/infokum

MRI Image Classification Analysis of Brain Cancer Using ResNet18 and VGG16 Deep Learning Architectures

Yudha Brema Agriva Sembiring[1], Evta Indra[2]
Information System, Faculty of Science and Technology, Universitas Prima Indonesia, North Sumatra, Indonesia

Article Info	ABSTRACT
Keywords: Image Classification, Deep Learning, ResNet18, VGG16	Deep learning approaches, particularly Convolutional Neural Networks (CNNs), have proven effective in medical image processing. Two prominent CNN architectures are VGG16 and ResNet18. Previous research has shown that ResNet50 and VGG16 have been used in brain tumor classification with varying accuracy. This study aims to systematically compare the performance of ResNet18 and VGG16 in brain cancer MRI image classification, considering accuracy, computational efficiency, and model generalization capabilities. The results show that the ResNet18 model with pretrained weights achieved the highest accuracy of 97.43%, excelling in detecting all categories of brain tumors with minimal error. In contrast, the VGG16 model trained from scratch performed the lowest with an accuracy of only 63.09%, having significant difficulty distinguishing between classes.
This is an open access article under theCC BY-NClicense	**Corresponding Author:** Evta Indra, Information System, Faculty of Scienty and Technology , Prima University, North Sumatra, evtaindra@unprimdn.ac.id

INTRODUCTION

Brain cancer is one of the most dangerous and complex types of cancer due to its aggressive growth and location, which affects vital body functions. To this end, medical imaging technologies such as magnetic resonance imaging (MRI) have become the standard for non-invasive and high-precision detection and visualization of brain tumors. MRI provides excellent images of the brain's soft tissues, enabling the identification of abnormalities such as gliomas, meningiomas, and pituitary tumors with high accuracy.

Manual magnetic resonance imaging (MRI) performed by radiologists requires high expertise and is a time-consuming process. Radiologists' assessments can also be influenced by subjectivity and interobserver variability, especially when the data volume is large and the tumor is highly complex. Therefore, it is necessary to develop a machine learning-based system to assist in the rapid and accurate classification of medical images.

Previous research using deep learning approaches, particularly Convolutional Neural Networks (CNN), has been widely used in medical image processing due to its ability to automatically extract spatial and textural features from raw data. Two prominent CNN architectures in the context of medical image classification are VGG16 and ResNet50. VGG16 has a consistent and simple layered structure, making it easy to analyze and implement, and has proven effective in general and medical image classification tasks.

Several studies have demonstrated the effectiveness of the ResNet18 and VGG16 architectures in brain tumor classification. In a study by (T. Agustin., 2024) the use of the ResNet18 and VGG16 architectures successfully achieved an accuracy of 88% in multi-class brain tumor classification. Meanwhile, a study by (M. B. Kurniawan, and E. Utami., 2025) showed that VGG16 recorded an accuracy of 94.93% in the classification of four types of brain tumors. Therefore, researchers will conduct a study entitled Analysis of Brain Cancer MRI Image Classification Using RestNet18 and VGG16 Deep Learning Architectures. Systematically comparing the performance of both directly in brain cancer MRI image classification by considering accuracy, computational efficiency, and model generalization capabilities.

This study aims to analyze the performance of the ResNet18 architecture in classifying MRI images of brain cancer and the performance of the VGG16 architecture in classifying MRI images of brain cancer. The two architectures are compared to determine the optimal model based on accuracy and computational efficiency. The results of this study are expected to contribute to improving early diagnosis of brain tumors with higher accuracy and provide new insights into the use of ResNet18 and VGG16 methods in classification.

METHOD

This study is a quantitative study with a comparative experimental approach. This study aims to compare the performance of two deep learning architectures, namely ResNet18 and VGG16, in the classification of MRI images of brain cancer. The study was conducted in silico (computer-based) using MRI image datasets from open sources.

The dataset used in this study is the Brain Tumor MRI Dataset from the Open Source Kaggle site (MIT Sartaj Bhuvaji) which contains four types of brain tumors: glioma, meningioma, and pituitary. ResNet18 is an 18-layer CNN architecture that uses residual connections to address the degradation problem of deep network training. ResNet allows models to maintain high accuracy without losing performance as the number of layers increases. VGG16 is a 16-layer CNN model that relies on small-sized (3x3) convolutions repeatedly. This model is known for its simplicity in architecture and its consistency across a variety of image recognition tasks.

RESULTS AND DISCUSSION

Leveraging the capabilities of ResNet18 and VGG16—two Convolutional Neural Network (CNN) architectures proven to be powerful in image classification and hierarchical feature extraction—this study attempts to build a model capable of automatically classifying brain cancer types. By comparing the performance of these two deep learning architectures, this study also aims to provide insight into which model is more optimal for the task of classifying brain cancer MRI images.

In processing lung cancer classification data in this study, a data processing flow is required to ensure good accuracy for the model used. Figure 3.1 illustrates the data processing flow in this study:

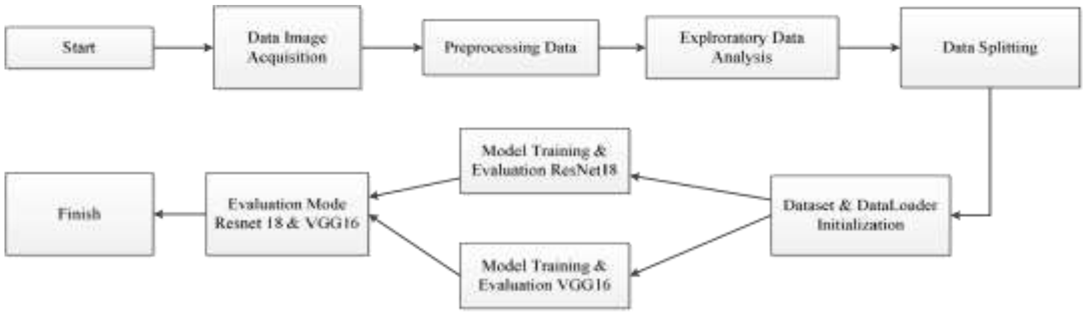
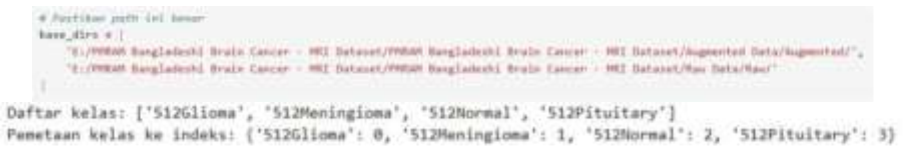

Start
Data Image Acquisition
Preprocessing Data
Exploratory Data Analysis
Data Splitting
Dataset & DataLoader Initialization
Model Training & Evaluation ResNet18
Model Training & Evaluation VGG16
Evaluation Mode Resnet 18 & VGG16
Finish

Figure 1 Data Processing Flow

Data acquisition

Data acquisition is a fundamental procedure carried out by researchers to collect relevant data. In this study, the data used were a total of 7509 image samples with 4 classes, namely '512Glioma': 0, '512Meningioma': 1, '512Normal': 2, '512Pituitary': 3 sourced from the open source data provider Bangladeshi Brain Cancer[11]. As illustrated in Figure 3.2 presented below:



```
# Pastikan path ini benar
base_dirs = [
    "E:/PMRAM Bangladeshi Brain Cancer - MRI Dataset/PMRAM Bangladeshi Brain Cancer - MRI Dataset/Augmented Data/Augmented/",
    "E:/PMRAM Bangladeshi Brain Cancer - MRI Dataset/PMRAM Bangladeshi Brain Cancer - MRI Dataset/Raw Data/Raw"
]
```

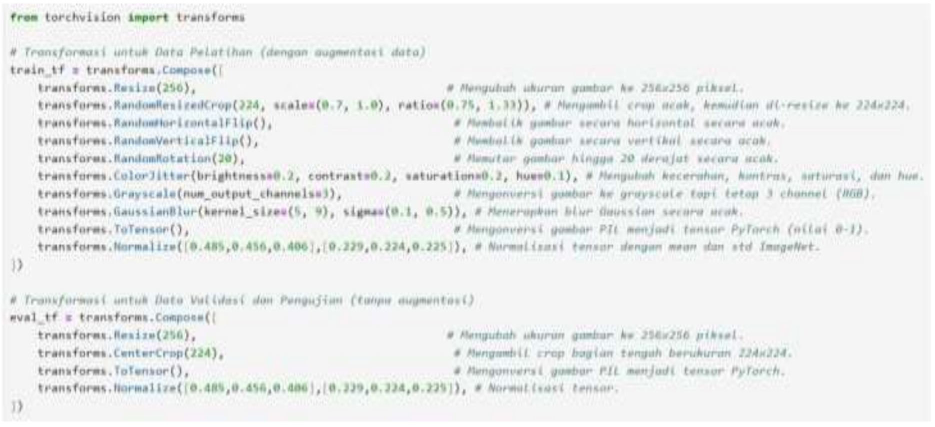
```
Daftar kelas: ['512Glioma', '512Meningioma', '512Normal', '512Pituitary']
Pemetaan kelas ke indeks: {'512Glioma': 0, '512Meningioma': 1, '512Normal': 2, '512Pituitary': 3}
```

Figure 2 Brain Cancer Classification Dataset

Preprocessing Data

The obtained data then proceeds to the preprocessing stage, where the dataset is prepared for the development and training of the machine learning model. At this point, the primary goal is to improve the quality of the dataset. The subsequent procedure is as follows:

Image Transform standardizes all images to a uniform 256 x 256 size and a format suitable for the model while enriching the dataset with data augmentation techniques.:

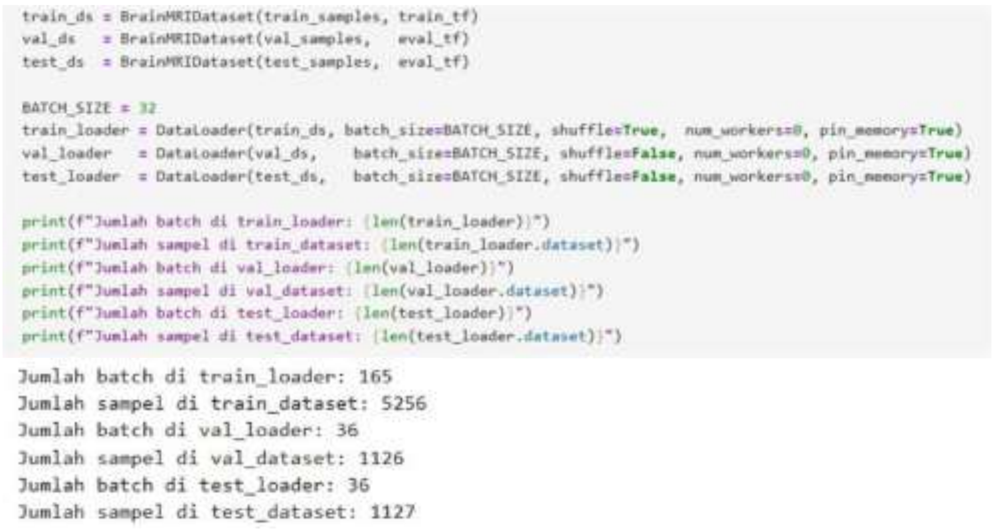


```
from torchvision import transforms

# Transformasi untuk Data Pelatihan (dengan augmentasi data)
train_tf = transforms.Compose([
    transforms.Resize(256),                                    # Mengubah ukuran gambar ke 256x256 piksel.
    transforms.RandomResizedCrop(224, scale=(0.7, 1.0), ratio=(0.75, 1.33)), # Mengambil crop acak, kemudian di-resize ke 224x224.
    transforms.RandomHorizontalFlip(),                          # Membalik gambar secara horizontal secara acak.
    transforms.RandomVerticalFlip(),                            # Membalik gambar secara vertikal secara acak.
    transforms.RandomRotation(20),                              # Memutar gambar hingga 20 derajat secara acak.
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1), # Mengubah kecerahan, kontras, saturasi, dan hue.
    transforms.Grayscale(num_output_channels=3),                # Mengonversi gambar ke grayscale tapi tetap 3 channel (RGB).
    transforms.GaussianBlur(kernel_size=(5, 9), sigma=(0.1, 0.5)), # Menerapkan blur Gaussian secara acak.
    transforms.ToTensor(),                                      # Mengonversi gambar PIL menjadi tensor PyTorch (nilai 0-1).
    transforms.Normalize([0.485,0.456,0.406],[0.229,0.224,0.225]), # Normalisasi tensor dengan mean dan std ImageNet.
])

# Transformasi untuk Data Validasi dan Pengujian (tanpa augmentasi)
eval_tf = transforms.Compose([
    transforms.Resize(256),                                    # Mengubah ukuran gambar ke 256x256 piksel.
    transforms.CenterCrop(224),                                 # Mengambil crop bagian tengah berukuran 224x224.
    transforms.ToTensor(),                                      # Mengonversi gambar PIL menjadi tensor PyTorch.
    transforms.Normalize([0.485,0.456,0.406],[0.229,0.224,0.225]), # Normalisasi tensor.
])
```

Figure 3 Image Transformation

Data Loader Initialization

At this stage, the dataset and data loader are initialized using previously divided data (training, validation, test) and defined image transformations.



```
train_ds = BrainMRIDataset(train_samples, train_tf)
val_ds   = BrainMRIDataset(val_samples,   eval_tf)
test_ds  = BrainMRIDataset(test_samples,  eval_tf)

BATCH_SIZE = 32
train_loader = DataLoader(train_ds, batch_size=BATCH_SIZE, shuffle=True,  num_workers=0, pin_memory=True)
val_loader   = DataLoader(val_ds,    batch_size=BATCH_SIZE, shuffle=False, num_workers=0, pin_memory=True)
test_loader  = DataLoader(test_ds,   batch_size=BATCH_SIZE, shuffle=False, num_workers=0, pin_memory=True)

print(f"Jumlah batch di train_loader: {len(train_loader)}")
print(f"Jumlah sampel di train_dataset: {len(train_loader.dataset)}")
print(f"Jumlah batch di val_loader: {len(val_loader)}")
print(f"Jumlah sampel di val_dataset: {len(val_loader.dataset)}")
print(f"Jumlah batch di test_loader: {len(test_loader)}")
print(f"Jumlah sampel di test_dataset: {len(test_loader.dataset)}")
```

```
Jumlah batch di train_loader: 165
Jumlah sampel di train_dataset: 5256
Jumlah batch di val_loader: 36
Jumlah sampel di val_dataset: 1126
Jumlah batch di test_loader: 36
Jumlah sampel di test_dataset: 1127
```

Figure 4 Data Loader Initialization

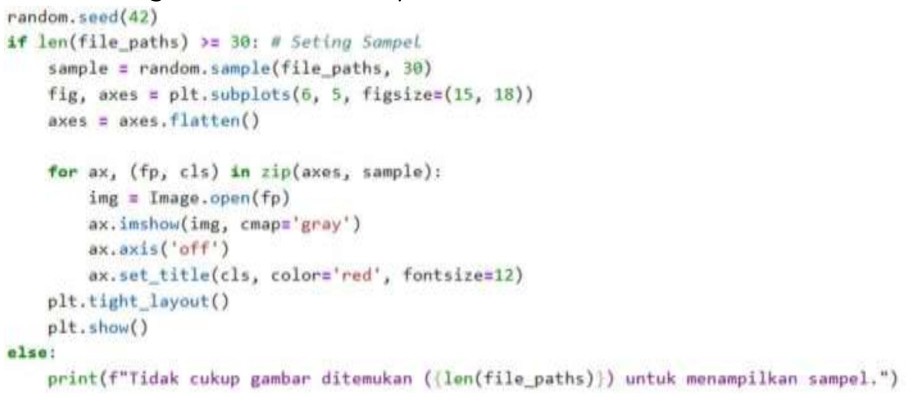
Figure 4 illustrates data partitioning statistics that are important in the context of developing machine learning or deep learning models. The training data (train_dataset) consists of 5,256 samples divided into 165 batches, which are used to "teach" the model to recognize patterns. Next, there is the validation data (val_dataset) consisting of 1,126 samples, divided into 36 batches; this data plays a crucial role in evaluating the model's performance during the training process and assists in parameter tuning to prevent overfitting. For the final evaluation after the model has been trained, the test data (test_dataset) consists of 1,127 samples, also divided into 36 batches.

Explroratory Data Analysis

This stage includes the initial stage of exploratory data analysis (EDA), which requires examining the distribution pattern of lung cancer cases. The data exploration carried out is as follows:

MRI Brain Image Samples from the Dataset

After carrying out the previous stages, the researcher presents the results of a random selection of 10 MRI brain images from a data set that includes three types of brain tumors, namely Glioma, Meningioma, and Pituitary.



```
random.seed(42)
if len(file_paths) >= 30: # Seting Sampel
    sample = random.sample(file_paths, 30)
    fig, axes = plt.subplots(6, 5, figsize=(15, 18))
    axes = axes.flatten()

    for ax, (fp, cls) in zip(axes, sample):
        img = Image.open(fp)
        ax.imshow(img, cmap='gray')
        ax.axis('off')
        ax.set_title(cls, color='red', fontsize=12)
    plt.tight_layout()
    plt.show()
else:
    print(f"Tidak cukup gambar ditemukan ({len(file_paths)}) untuk menampilkan sampel.")
```

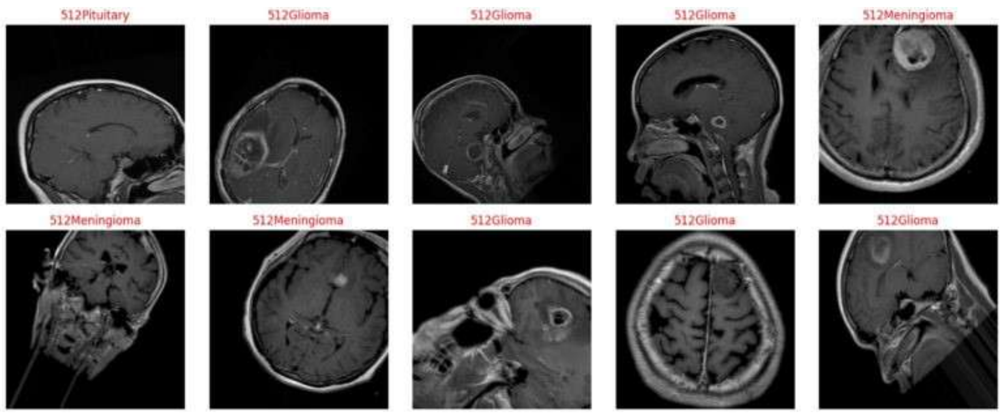
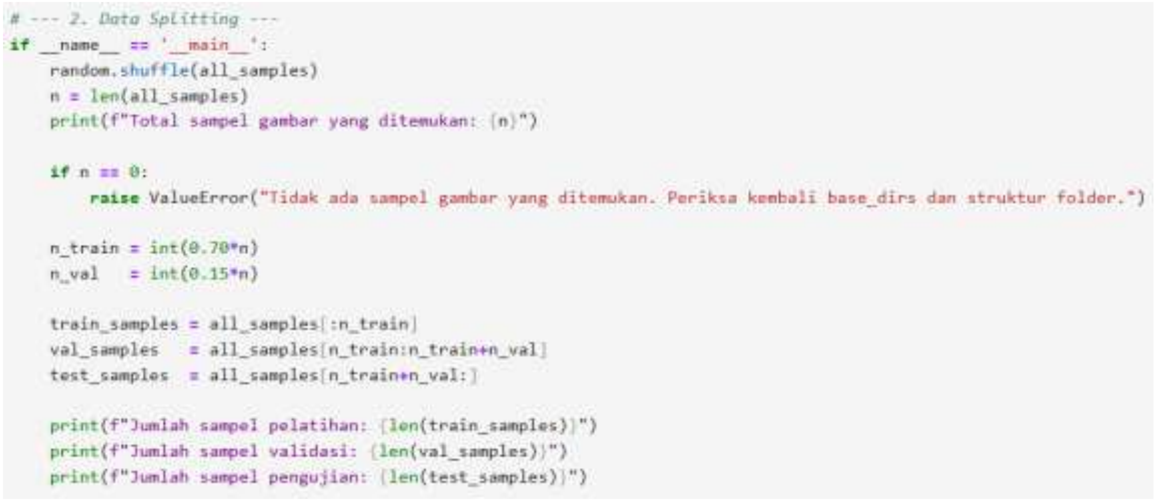


Figure 5 MRI Image of the Brain

Based on the visualization results, Figure 3.5 displays a collection of MRI images of the brain classified as "Pituitary" (referring to a pituitary gland tumor), "Glioma" (a tumor originating from glial cells), "Meningioma" (a tumor originating from the meninges), and "Normal" (indicating no significant abnormalities).

Data Splitting

The data splitting stage is an essential phase in this research methodology. It aims to allocate the entire brain MRI image dataset into strategically distributed subsets: a training set, a validation set, and a test set.



```
# --- 2. Data Splitting ---
if __name__ == '__main__':
    random.shuffle(all_samples)
    n = len(all_samples)
    print(f"Total sampel gambar yang ditemukan: {n}")

    if n == 0:
        raise ValueError("Tidak ada sampel gambar yang ditemukan. Periksa kembali base_dirs dan struktur folder.")

    n_train = int(0.70*n)
    n_val   = int(0.15*n)

    train_samples = all_samples[:n_train]
    val_samples   = all_samples[n_train:n_train+n_val]
    test_samples  = all_samples[n_train+n_val:]

    print(f"Jumlah sampel pelatihan: {len(train_samples)}")
    print(f"Jumlah sampel validasi: {len(val_samples)}")
    print(f"Jumlah sampel pengujian: {len(test_samples)}")
```

Figure 6 Distribution of Training, Validation, and Testing Data

Figure 6 shows a crucial stage in data preprocessing for machine learning, namely dataset partitioning. The entire collection of image samples (all_samples) is randomly shuffled to ensure an unbiased data distribution and prevent the original data order from influencing the model. The total number of samples (n) is calculated, and if no samples are found, the program will terminate execution with an error message.

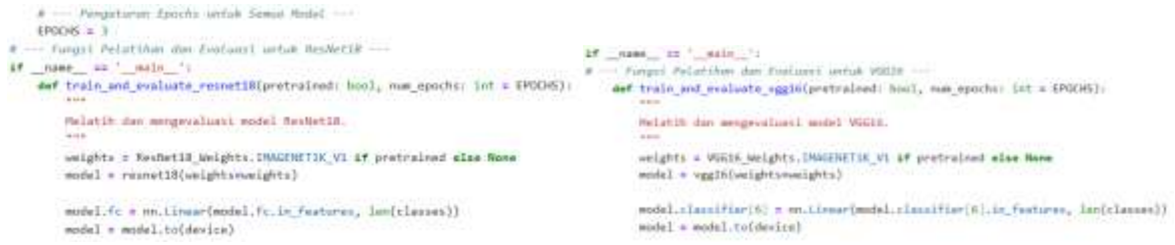
The dataset is divided into three subsets: 70% for training (train_samples), 15% for validation (val_samples), and 15% for testing (test_samples) using an integer calculation of the percentage of the total samples. This division process is carried out through array slicing where the data is taken from the starting index to the ending index that has been determined.

Resnet18 and VGG16 Modeling

This research uses the Resnet18 and VGG16 methods using the Python library, which can be described as follows:

1. Resnet18 and VGG16 Model Initialization

Model initialization involves preparing the model with initial values (random values) before training it with data to ensure an effective and stable learning process.



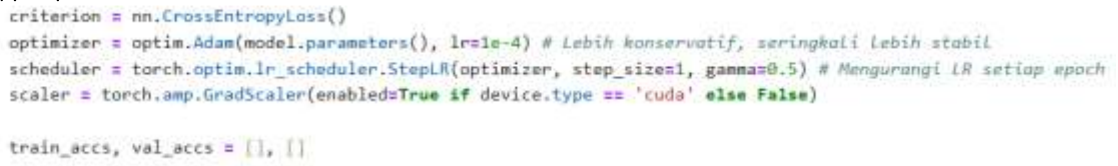
```python
# --- Pengaturan Epochs untuk Semua Model ---
EPOCHS = 3
# --- Fungsi Pelatihan dan Evaluasi untuk ResNet18 ---
if __name__ == '__main__':
    def train_and_evaluate_resnet18(pretrained: bool, num_epochs: int = EPOCHS):
        """
        Melatih dan mengevaluasi model ResNet18.
        """
        weights = ResNet18_Weights.IMAGENET1K_V1 if pretrained else None
        model = resnet18(weights=weights)

        model.fc = nn.Linear(model.fc.in_features, len(classes))
        model = model.to(device)
```

```python
if __name__ == '__main__':
# --- Fungsi Pelatihan dan Evaluasi untuk VGG16 ---
    def train_and_evaluate_vgg16(pretrained: bool, num_epochs: int = EPOCHS):
        """
        Melatih dan mengevaluasi model VGG16.
        """
        weights = VGG16_Weights.IMAGENET1K_V1 if pretrained else None
        model = vgg16(weights=weights)

        model.classifier[6] = nn.Linear(model.classifier[6].in_features, len(classes))
        model = model.to(device)
```

Figure 7 Initialization of ResNet18 and VGG16 Models

In Figure 7, the pretrained and scratch sections provide the option to choose whether the model will use pre-trained weights (pretrained=True) or be trained from scratch (pretrained=False, called "from scratch"). If pretrained=True, a model like ResNet18 or VGG16 will load the weights trained on a large dataset like ImageNet, giving it a baseline ability to recognize common visual patterns. This pretrained=True configuration speeds up training and typically results in better accuracy, especially when training data is limited. However, if pretrained=False, the model starts with random weights and must learn everything from scratch, which requires more data, time, and typically results in lower performance when data is scarce.

2. Hyperparameter Definition



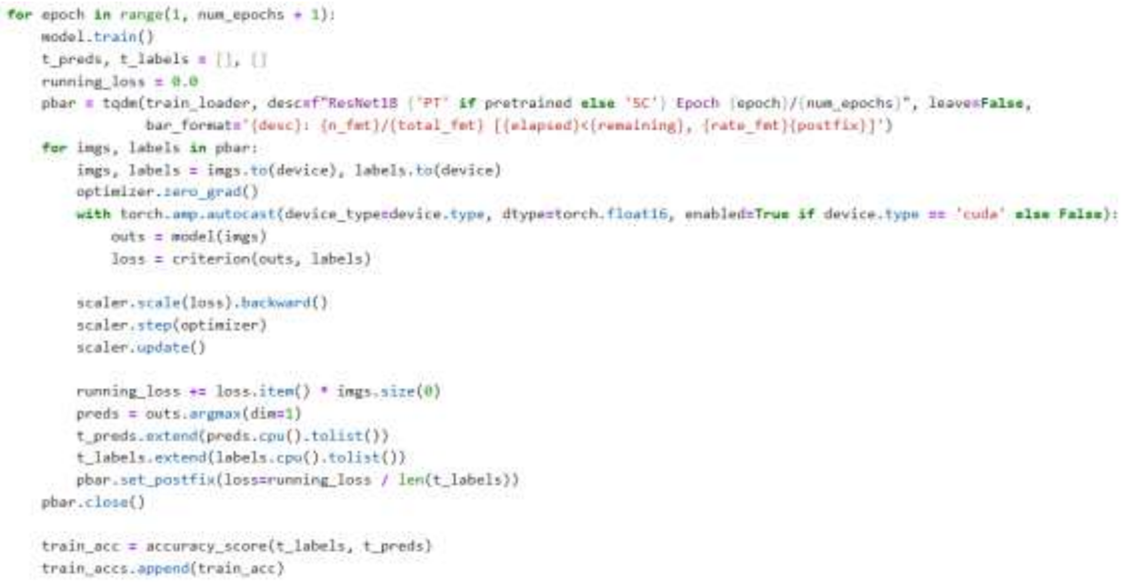
```python
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-4) # Lebih konservatif, seringkali lebih stabil
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=1, gamma=0.5) # Mengurangi LR setiap epoch
scaler = torch.amp.GradScaler(enabled=True if device.type == 'cuda' else False)

train_accs, val_accs = [], []
```

Figure 8 Hyperparameter Settings

Figure 8 shows the process for setting important components in the deep learning model training process. First, criterion = nn.CrossEntropyLoss() specifies the loss function used for the multiclass classification task. optimizer = optim.Adam(model.parameters(), lr=1e-4) defines the optimization algorithm using Adam with a conservative learning rate of 0.0001, which is generally more stable. scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=1, gamma=0.5) is used to gradually decrease the learning rate every epoch by a reduction factor of 0.5. scaler = torch.amp.GradScaler(enabled=True if device.type == 'cuda' else False) enables automatic mixed precision to accelerate the training process using GPU (CUDA). Train_accs and val_accs are two empty lists used to store accuracy values for the training and validation data during the training process.

3. Training Loop

The training loop is the stage where the model learns from the data through iterations per epoch. At each epoch, the model is set to training mode, and the image data and labels are processed in batches.



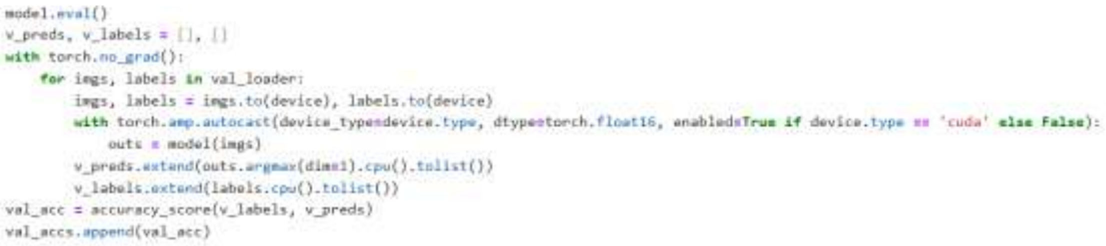
```
for epoch in range(1, num_epochs + 1):
    model.train()
    t_preds, t_labels = [], []
    running_loss = 0.0
    pbar = tqdm(train_loader, desc=f"ResNet18 {'PT' if pretrained else 'SC'} Epoch {epoch}/{num_epochs}", leave=False,
                bar_format='{desc}: {n_fmt}/{total_fmt} [{elapsed}<{remaining}, {rate_fmt}{postfix}]')
    for imgs, labels in pbar:
        imgs, labels = imgs.to(device), labels.to(device)
        optimizer.zero_grad()
        with torch.amp.autocast(device_type=device.type, dtype=torch.float16, enabled=True if device.type == 'cuda' else False):
            outs = model(imgs)
            loss = criterion(outs, labels)

        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()

        running_loss += loss.item() * imgs.size(0)
        preds = outs.argmax(dim=1)
        t_preds.extend(preds.cpu().tolist())
        t_labels.extend(labels.cpu().tolist())
        pbar.set_postfix(loss=running_loss / len(t_labels))
    pbar.close()

    train_acc = accuracy_score(t_labels, t_preds)
    train_accs.append(train_acc)
```

Figure 9 Training Loop Settings

At each epoch, the model is set to training mode (model.train()), then iterates over the training data. Image data and labels are transferred to the device, and the training process uses mixed precision (torch.amp.autocast) for efficiency when using a GPU (CUDA). After predictions are made and the loss is calculated, backpropagation is performed and the model weights are updated using a scaler for stability under mixed precision. Training accuracy is calculated using accuracy_score and stored in train_accs. Training progress is displayed in real time using tqdm.

4. Validation Loop

The validation loop is performed after training to evaluate model performance on untrained data.



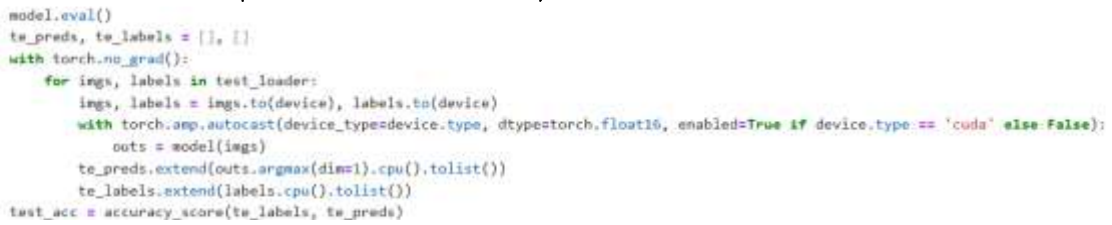
```
model.eval()
v_preds, v_labels = [], []
with torch.no_grad():
    for imgs, labels in val_loader:
        imgs, labels = imgs.to(device), labels.to(device)
        with torch.amp.autocast(device_type=device.type, dtype=torch.float16, enabled=True if device.type == 'cuda' else False):
            outs = model(imgs)
        v_preds.extend(outs.argmax(dim=1).cpu().tolist())
        v_labels.extend(labels.cpu().tolist())
val_acc = accuracy_score(v_labels, v_preds)
val_accs.append(val_acc)
```

Figure 10 Validation Loop Settings

Figure 10 shows the model evaluation process on the validation dataset (val_loader). It begins by setting the model to evaluation mode (model.eval()) to ensure consistent behavior, then iterates over each batch of data without calculating gradients (torch.no_grad()) for efficiency. The images and labels are transferred to the appropriate compute device, and the model performs inference. The model predictions and original labels are collected, and finally, accuracy is calculated using accuracy_score to measure the model's performance on data never seen during training.

5. Test Set Process

The test set is a subset of the entire dataset that is intentionally kept separate and never used during the training or validation processes. Its purpose is to provide an unbiased evaluation of the model's performance on entirely new data.



```
model.eval()
te_preds, te_labels = [], []
with torch.no_grad():
    for imgs, labels in test_loader:
        imgs, labels = imgs.to(device), labels.to(device)
        with torch.amp.autocast(device_type=device.type, dtype=torch.float16, enabled=True if device.type == 'cuda' else False):
            outs = model(imgs)
        te_preds.extend(outs.argmax(dim=1).cpu().tolist())
        te_labels.extend(labels.cpu().tolist())
test_acc = accuracy_score(te_labels, te_preds)
```

Figure 11 Test Set Configuration

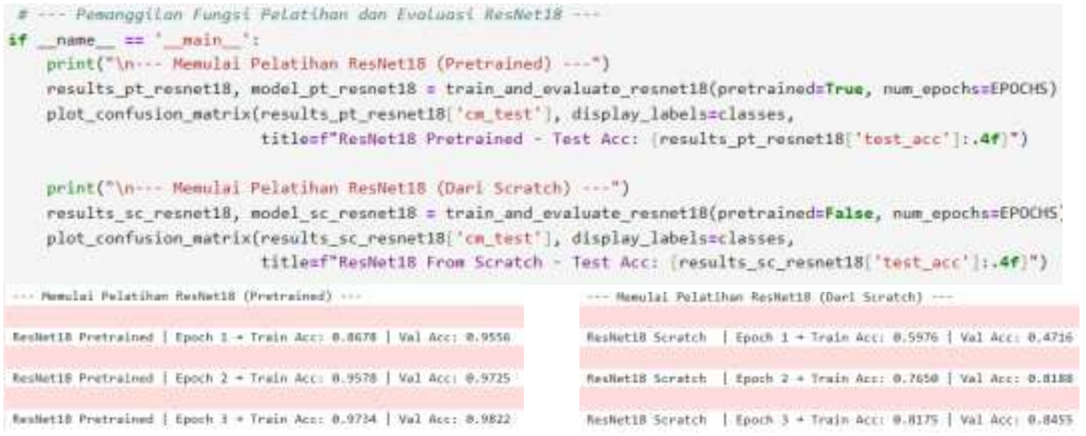
In Figure 11, model evaluation after the training process is complete begins by setting the model to evaluation mode via model.eval(), which disables features such as dropout and batch normalization to ensure more stable evaluation results. Evaluation is performed without calculating gradients (with torch.no_grad()), making the process faster and more memory efficient. The model then generates predictions (outs), which are converted to predicted classes using argmax and collected into the te_preds list. The actual labels are also collected into te_labels. After all data is tested, accuracy is calculated using accuracy_score(te_labels, te_preds), which provides a metric of the model's performance on the test data.

Model Evaluation

The evaluation models used were ResNet18 and VGG16 to assess the ability of these two CNN architectures to classify medical images by comparing the effectiveness of training from scratch versus using pretrained weights. The results showed significant advantages of transfer learning.

1. ResNet18

ResNet18 stands for Residual Network, with 18 layers. Its main feature is the use of "skip connections" or "residual connections," which allow gradients to flow more easily through deep layers, addressing the problem of vanishing gradients in very deep networks.



```
# --- Pemanggilan Fungsi Pelatihan dan Evaluasi ResNet18 ---
if __name__ == '__main__':
    print("\n--- Memulai Pelatihan ResNet18 (Pretrained) ---")
    results_pt_resnet18, model_pt_resnet18 = train_and_evaluate_resnet18(pretrained=True, num_epochs=EPOCHS)
    plot_confusion_matrix(results_pt_resnet18['cm_test'], display_labels=classes,
                          title=f"ResNet18 Pretrained - Test Acc: {results_pt_resnet18['test_acc']:.4f}")

    print("\n--- Memulai Pelatihan ResNet18 (Dari Scratch) ---")
    results_sc_resnet18, model_sc_resnet18 = train_and_evaluate_resnet18(pretrained=False, num_epochs=EPOCHS
    plot_confusion_matrix(results_sc_resnet18['cm_test'], display_labels=classes,
                          title=f"ResNet18 From Scratch - Test Acc: {results_sc_resnet18['test_acc']:.4f}")
```

```
--- Memulai Pelatihan ResNet18 (Pretrained) ---

ResNet18 Pretrained | Epoch 1 → Train Acc: 0.8678 | Val Acc: 0.9556

ResNet18 Pretrained | Epoch 2 → Train Acc: 0.9578 | Val Acc: 0.9725

ResNet18 Pretrained | Epoch 3 → Train Acc: 0.9734 | Val Acc: 0.9822
```

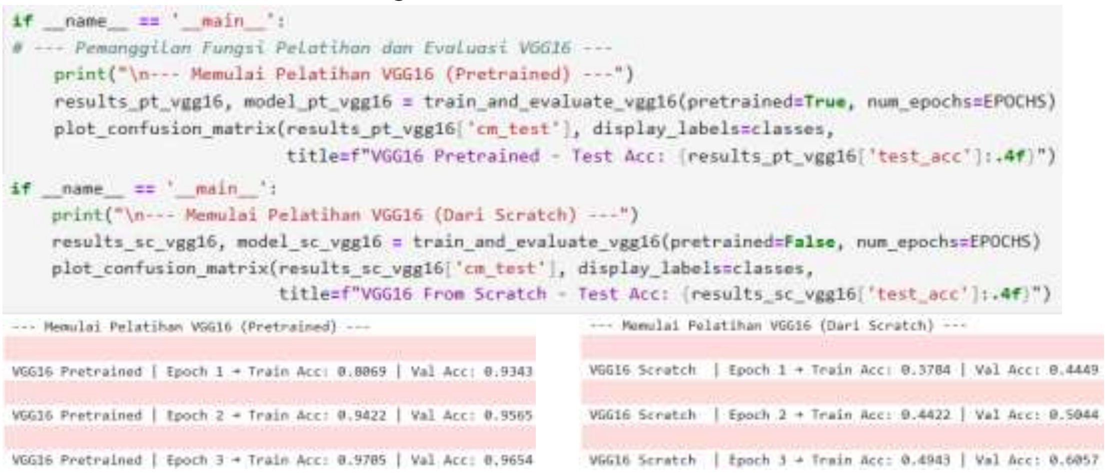
```
--- Memulai Pelatihan ResNet18 (Dari Scratch) ---

ResNet18 Scratch   | Epoch 1 → Train Acc: 0.5976 | Val Acc: 0.4716

ResNet18 Scratch   | Epoch 2 → Train Acc: 0.7650 | Val Acc: 0.8188

ResNet18 Scratch   | Epoch 3 → Train Acc: 0.8175 | Val Acc: 0.8455
```

Figure 12 ResNet18 Evaluation Model

Figure 12 presents a comparison of the training results of the ResNet18 model over three epochs. Figure 1 shows the performance of the ResNet18 model trained using pretrained weights, where training accuracy increased from 0.8678 to 0.9734 and validation accuracy from 0.9556 to 0.9822. Figure 2 shows the results of ResNet18 training from scratch, with lower training accuracy (from 0.5976 to 0.8175) and lower validation accuracy (from 0.4716 to 0.8455). This comparison clearly indicates that the use of pretrained weights provides significant benefits in model accuracy and convergence for classification results.

2. VGG16

VGG16 is a Visual Geometry Group Network with 16 layers. VGG is known for its simple and uniform architecture, using recurrent 3x3 convolutional blocks.



```
if __name__ == '__main__':
# --- Pemanggilan Fungsi Pelatihan dan Evaluasi VGG16 ---
    print("\n--- Memulai Pelatihan VGG16 (Pretrained) ---")
    results_pt_vgg16, model_pt_vgg16 = train_and_evaluate_vgg16(pretrained=True, num_epochs=EPOCHS)
    plot_confusion_matrix(results_pt_vgg16['cm_test'], display_labels=classes,
                          title=f"VGG16 Pretrained - Test Acc: {results_pt_vgg16['test_acc']:.4f}")
if __name__ == '__main__':
    print("\n--- Memulai Pelatihan VGG16 (Dari Scratch) ---")
    results_sc_vgg16, model_sc_vgg16 = train_and_evaluate_vgg16(pretrained=False, num_epochs=EPOCHS)
    plot_confusion_matrix(results_sc_vgg16['cm_test'], display_labels=classes,
                         title=f"VGG16 From Scratch - Test Acc: {results_sc_vgg16['test_acc']:.4f}")
```

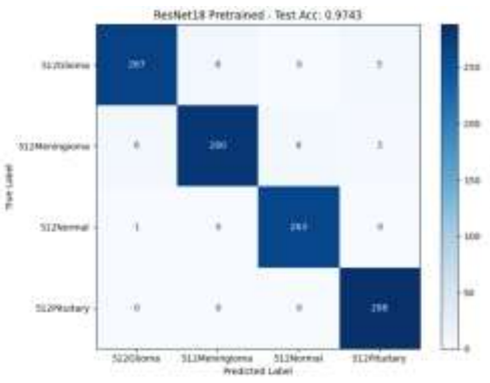
```
--- Memulai Pelatihan VGG16 (Pretrained) ---

VGG16 Pretrained | Epoch 1 → Train Acc: 0.8069 | Val Acc: 0.9343

VGG16 Pretrained | Epoch 2 → Train Acc: 0.9422 | Val Acc: 0.9565

VGG16 Pretrained | Epoch 3 → Train Acc: 0.9705 | Val Acc: 0.9654
```

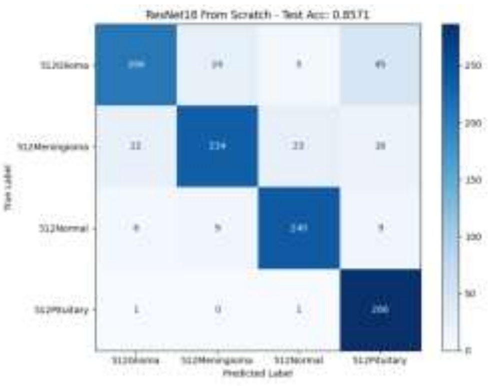
```
--- Memulai Pelatihan VGG16 (Dari Scratch) ---

VGG16 Scratch  | Epoch 1 → Train Acc: 0.3784 | Val Acc: 0.4449

VGG16 Scratch  | Epoch 2 → Train Acc: 0.4422 | Val Acc: 0.5044

VGG16 Scratch  | Epoch 3 → Train Acc: 0.4943 | Val Acc: 0.6057
```

Figure 13: VGG16 Model Evaluation

Figure 3.13 presents a comparison of the training results of the VGG16 model over three epochs. Figure 3 shows the performance of the VGG16 model trained using pretrained weights, where training accuracy increased from 0.8069 to 0.9705 and validation accuracy from 0.9343 to 0.9654. Figure 4 shows the results of VGG16 training from scratch, with lower training accuracy (from 0.3784 to 0.4943) and lower validation accuracy (from 0.3784 to 0.6057). This comparison clearly indicates that the use of pretrained weights provides significant benefits in model accuracy and convergence for classification results.

3. ResNet18 Confusion Matrix



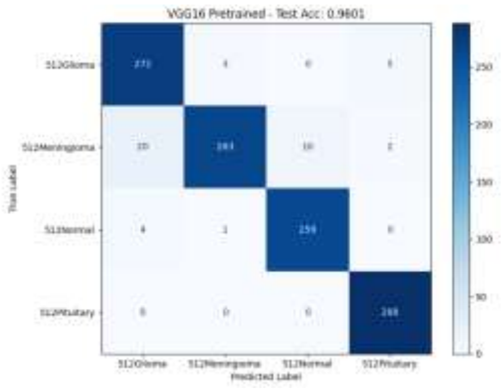


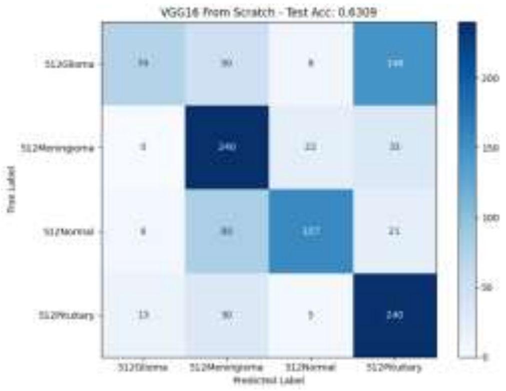
Pretrained ResNet18 Confusion Matrix   Scratch ResNet18 Confusion Matrix

Figure 14. ResNet18 Confusion Matrix

Figure 14 (pretrained ResNet18) has the highest accuracy of 97.43%, with highly accurate classification across all classes, significantly better than the second image (from-scratch ResNet18), which only achieved 85.71% and showed numerous misclassifications, particularly between the Glioma and Pituitary classes.

4. VGG16 Confusion Matrix





Pretrained VGG16 Confusion Matrix Scratch VGG16 Confusion Matrix

Figure 15. VGG16 Confusion Matrix

Figure 15 (pretrained VGG16) also shows high performance with an accuracy of 96.01%, while the fourth image (from-scratch VGG16) has the lowest accuracy of 63.09% and numerous mispredictions. almost all classes. Models using pretrained weights performed significantly better than models trained from scratch. Scratch, with pretrained ResNet18 as the best model in the final results, provided a strong foundation for the model to understand important visual features, even with domain-specific datasets like brain MRIs.

CONCLUSION

In this study, the best model was ResNet18 with pretrained weights, achieving the highest accuracy of 97.43% and demonstrating highly accurate classification across all brain tumor classes. This model excelled in detecting all categories with minimal error. The lowest-performing model was VGG16, trained from scratch, with an accuracy of only 63.09%. This model struggled to differentiate between classes, with high error rates across almost all categories.

Table 1 Deep Learning Performance Comparison

Aspects	(ResNet18 PT)	(ResNet18 FS)	(VGG16 PT)	(VGG16 FS)
Pretrained/ From Scratch	Pretrained	From Scratch	Pretrained	From Scratch
Architecture	ResNet18	ResNet18	VGG16	VGG16
Accuracy	0.9743	0.8571	0.9601	0.6309
Performance	Very Good	Fair Good	Very Good	Poor
Classification Error	A little	Much	A little	Much

REFERENCE

[1] S. Agarwal, D. Jain, S. Gupta, S. Sawhney, and Y. Mittal, "Brain Tumor Detection and Classification Using Deep Learning," *Proc. - IEEE 2023 5th Int. Conf. Adv. Comput. Commun. Control Networking, ICAC3N 2023*, pp. 635–640, 2023, doi: 10.1109/ICAC3N60023.2023.10541584.

[2] J. Seetha and S. S. Raja, "Brain tumor classification using Convolutional Neural Networks," *Biomed. Pharmacol. J.*, vol. 11, pp. 1457–1461, Sep. 2018, doi: 10.13005/bpj/1511.

[3] R. Andre, B. Wahyu, and R. Purbaningtyas, "Klasifikasi Tumor Otak Menggunakan Convolutional Neural Network Dengan Arsitektur Efficientnet-B3," *J. IT*, vol. 11, no. 3, pp. 55–59, 2021, [Online]. Available: https://jurnal.umj.ac.id/index.php/just-it/index

[4] M. L. Septipalan, M. S. Hibrizi, N. Latifah, R. Lina, and F. Bimantoro, "Klasifikasi Tumor Otak Menggunakan CNN Dengan Arsitektur ResNet18," *Semin. Nas. Teknol. Sains*, vol. 3, no. 1, pp. 103–108, 2024, doi: 10.29407/stains.v3i1.4357.

[5] D. Lizard, S. Dimara, S. W. Putri, and R. Amelia, "Penerapan Convolutional Neural Network (CNN) dalam Klasifikasi Citra MRI untuk Deteksi Tumor Otak Manusia," vol. 4, no. 2, pp. 70–77, 2023, doi: 10.31284/j.kernel.2023.v4i2.6960.

[6] K. Amalia, R. Magdalena, and S. Saidah, "Klasifikasi Penyakit Tumor Otak Pada Citra Mri Menggunakan Metode CNN," *e-Proceeding Eng.*, vol. 8, no. 6, pp. 3247–3254, 2022.

[7] T. A. Mutiara and Q. N. Azizah, "Klasifikasi Tumor Otak Menggunakan Ekstraksi Fitur HOG dan Support Vector Machine," *J. Infortech*, vol. 4, no. 1, pp. 45–50, 2022, [Online]. Available: https://ejournal.bsi.ac.id/ejurnal/index.php/infortech/article/view/12813

[8] K. N. Qodri, "Analisis Perbandingan Klasifikasi Tumor Otak Menggunakan Deep Learning," vol. 1, pp. 1–6, 2024.

[9] T. Agustin, "ANALISIS PERFORMA MODEL DEEP LEARNING VGG16 DAN RESNET DALAM KLASIFIKASI JENIS TUMOR OTAK," no. November, pp. 136–147, 2024.

[10] T. Otak, M. B. Kurniawan, and E. Utami, "Perbandingan Kinerja ResNet18 , VGG16 , dan MobileNetV2 Performance Comparison of ResNet18 , VGG16 , and MobileNetV2 for Brain Tumor Classification on MRI Images," vol. 14, pp. 767–777, 2025.

[11] "Brain Tumor Classification (MRI)." Accessed: Apr. 29, 2025. [Online]. Available: https://www.kaggle.com/datasets/sartajbhuvaji/brain-tumor-classification-mri

[12] A. Saleh, R. Sukaik, and S. S. Abu-Naser, "Brain tumor classification using deep learning," in *Proceedings - 2020 International Conference on Assistive and Rehabilitation Technologies, iCareTech 2020*, Institute of Electrical and Electronics Engineers Inc., Aug. 2020, pp. 131–136. doi: 10.1109/iCareTech49914.2020.00032.

[13] D. Candra, G. Wibisono, M. Ayu, and M. Afrad, "Transfer Learning model Convolutional Neural Network menggunakan VGG-16 untuk Klasifikasi Tumor Otak pada Citra Hasil MRI," *LEDGER J. Inform. Inf. Technol.*, vol. 3, no. 1, pp. 11–18, 2024.