Tree Species Classification Using Deep Learning Architectures

Identifying Tree Species Using Leaf Images (Flavia Dataset)

**Project Description**

The goal of this project is to classify tree species using images of their leaves. The dataset used for this task is the Flavia Dataset, which contains a variety of leaf images from different tree species. The project involves implementing a ResNet architecture from scratch and utilizing Xception and DenseNet as pre-trained models, applying transfer learning techniques to fine-tune them on the Flavia Dataset.

**ResNet (Residual Network)**

Introduction:

ResNet (He et al., 2015) introduced the concept of "residual learning" to overcome the vanishing gradient problem in deep networks. ResNet uses skip connections to improve learning in deeper architectures.

Key Characteristics:

- Residual Blocks: y = F(x) + x where F(x) is the transformation function.

- Deep architectures with varying depths (e.g., ResNet-18, ResNet-50, ResNet-152).

Pros:

- Solves vanishing gradient problems with skip connections.

- Enables training very deep networks.

- Excellent performance on image classification tasks.

Cons:

- High computational cost for deeper versions.

- Requires significant training time when built from scratch.

Reference**:**

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778. [Link](https://arxiv.org/abs/1512.03385)

**Xception (Extreme Inception)**

Introduction:

Xception (Chollet, 2017) is an improvement over Inception by using depthwise separable convolutions to reduce computational cost while maintaining performance.

Key Characteristics:

- Combines depthwise separable convolutions with residual connections.

- Based on the Inception architecture but replaces standard convolutions with depthwise separable convolutions.

Pros:

- Fewer parameters compared to ResNet.

- Works well with transfer learning on small datasets.

- Reduces computational cost compared to standard convolutional layers.

Cons:

- Requires larger datasets or good augmentation to generalize well.

- May not perform optimally on small datasets without transfer learning.

Reference:

Chollet, F. (2017). Xception: Deep Learning with Depthwise Separable Convolutions. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1251–1258. [Link](https://arxiv.org/abs/1610.02357)

**DenseNet (Densely Connected Convolutional Network)**

Introduction:

DenseNet (Huang et al., 2017) uses densely connected blocks, where each layer connects to all preceding layers.

Key Characteristics:

- Dense Connectivity: Improves gradient flow and reduces vanishing gradient issues.

- Fewer parameters compared to ResNet but better feature reuse.

Pros:

- Efficient parameter usage.

- Improves feature propagation and reuse.

- Strong performance on small and medium-sized datasets.

Cons:

- Increased memory usage due to dense connections.

- Can be computationally expensive for large images.

Reference**:**

Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely Connected Convolutional Networks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 4700–4708. [Link](https://arxiv.org/abs/1608.06993)

The performance of ResNet, Xception, and DenseNet depends on various factors, including the dataset's characteristics, preprocessing, and training conditions , So:  
**ResNet**

* Strengths**:** Excels on deep datasets with diverse features. Handles the vanishing gradient problem effectively. Works well even with relatively small datasets.
* Weaknesses**:**  High computational cost, especially for deeper versions like ResNet-101 or ResNet-152.
* Performance**:** ResNet is a strong baseline and typically achieves high accuracy, but may require more epochs to converge compared to lighter architectures.

**Xception**

* Strengths**:**  Efficient parameter usage. Outperforms many architectures on large datasets like ImageNet. Depthwise separable convolutions make it computationally lighter than ResNet.
* Weaknesses**:** Needs careful augmentation or larger datasets to generalize well.
* Performance**:**  Xception often performs better than ResNet for image classification tasks when transfer learning is applied, especially on datasets with clear and distinct features like Flavia.

**DenseNet**

* Strengths**:** Feature reuse leads to efficient learning with fewer parameters. Typically achieves higher accuracy on smaller datasets compared to ResNet and Xception.
* Weaknesses**:** High memory requirements and slower inference due to dense connectivity.
* Performance**:**  DenseNet often outperforms ResNet and Xception on tasks involving small-to-medium-sized datasets due to its efficient feature reuse. For Flavia, DenseNet might achieve slightly better accuracy than Xception if memory is not a constraint.

**for Flavia Dataset:**

* DenseNet**:**  Likely to give the best results due to its efficient learning and reuse of features. Ideal for datasets with fewer samples or when computational resources are sufficient.
* Xception**:**  A close second, particularly if computational efficiency during training is a priority.
* ResNet**:**  Reliable but may lag behind DenseNet and Xception in accuracy for this specific task.

Comparative Analysis:

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Strengths** | **Weaknesses** |
| ResNet | Handles very deep networks effectively. | High computational cost. |
| Xception | Efficient parameter usage. | Requires more data for generalization. |
| DenseNet | Encourages feature reuse. | High memory usage. |

**Models**

**ResNet**

Dataset Loading

1. load\_flavia\_dataset\_from\_csv Function:

- Reads the Flavia dataset from a CSV file containing image paths and labels.

- Ensures the dataset is loaded efficiently, with error handling for missing or corrupted images.

- Outputs:

- X: Array of preprocessed images (normalized).

- y: Integer-encoded labels.

- y\_onehot: One-hot encoded labels.

- class\_names: Unique class labels for mapping.

---

Preprocessing

Loading and Normalizing Images:

- Reads images from file paths specified in the CSV file.

- Resizes images to (224 \times 224) pixels using high-quality downsampling (LANCZOS).

- Converts images to RGB format, ensuring uniform three-channel inputs.

- Normalizes pixel values to the range \([0, 1]\) for stabilized training.

Label Handling:

- Extracts class labels from the CSV file.

- Converts integer class labels to one-hot encoded format using to\_categorical for compatibility with the categorical cross-entropy loss function.

Validation and Filtering:

- Validates file paths and ensures images are of acceptable formats using imghdr.

- Skips invalid or corrupted files and logs warnings during the preprocessing step.

Training-Test Split:

- Splits the dataset into training and testing subsets (80-20 ratio).

- Ensures stratified splitting to maintain balanced class distributions.

- Provides labels in both integer (y\_train, y\_test) and one-hot encoded (y\_train\_onehot, y\_test\_onehot) formats.

Data Augmentation:

- Applies random transformations to the training images to enhance dataset diversity.

- Techniques include random rotations, width/height shifts, horizontal flips, zooming, and shearing.

- Ensures image integrity is maintained using the nearest fill mode.

---

ResNet Implementation

Custom ResNet Architecture:

- Residual Blocks:

- Combines convolution layers with shortcut connections to combat vanishing gradients and enhance training of deep networks.

- Includes downsampling blocks to reduce spatial dimensions while increasing filter depth.

- Feature Extraction:

- Starts with an initial convolution layer and max-pooling to extract primary features.

- Progressively deeper layers capture more complex features.

- Batch Normalization:

- Normalizes feature maps after convolution layers to stabilize learning and improve convergence.

- Global Average Pooling (GAP):

- Reduces feature maps into a single vector for each channel, ensuring compact representation before classification.

- Dropout Regularization:

- Prevents overfitting by randomly dropping connections in the fully connected layers.

Output Layer:

- A fully connected softmax layer outputs probabilities for each class.

Compilation:

- Optimizer: Adam with a learning rate of (10^{-3}).

- Loss Function: Categorical cross-entropy for multi-class classification.

- Metrics: Accuracy.

---

Training

Callbacks:

- ReduceLROnPlateau: Dynamically reduces the learning rate if validation loss stagnates.

- EarlyStopping: Halts training early if validation accuracy fails to improve over 15 consecutive epochs.

Training Process:

- Uses the model.fit function with an 80-20 validation split.

- Logs metrics such as accuracy and loss for both training and validation sets.

---

Evaluation

Test Set Evaluation:

- model.evaluate calculates the final test loss and accuracy.

Predictions:

- model.predict generates class probabilities for the test set.

- Final predictions are obtained by taking the argmax of the probabilities.

Classification Report:

- Detailed metrics (precision, recall, F1-score) for each class are computed using classification\_report.

---

Visualizations

Training History:

- Plots training and validation accuracy/loss across epochs to evaluate model performance over time.

Confusion Matrix:

- Provides a heatmap showing the model's classification performance across all classes.

Precision-Recall Curve:

- Shows the tradeoff between precision and recall for each class, highlighting the model's ability to balance false positives and false negatives.

ROC Curve:

- Plots the true positive rate (TPR) vs. false positive rate (FPR) for each class.

- Computes the area under the curve (AUC) as an indicator of classification performance.

---

**Xception**

1. Dataset Loading

The load\_flavia\_dataset\_from\_csv function reads a dataset from a CSV file, processes the image data, and prepares it for model training. It:

- Loads image paths and class labels.

- Resizes images to (299, 299) (the expected input size for most models like Xception or DenseNet).

- Normalizes image pixel values to [0, 1].

- One-hot encodes the labels for multi-class classification.

2. Data Splitting

The dataset is divided into:

- Training + validation set (80% of the total dataset).

- Test set (20% of the total dataset).

The training + validation set is further split into:

- Training set (80% of the 80% subset).

- Validation set (20% of the 80% subset).

3. Data Augmentation

The ImageDataGenerator applies transformations to training images to artificially expand the dataset:

- Rotation, width/height shifts, horizontal/vertical flips, zoom, and shear augmentations.

- These transformations make the model more robust to variations in the data.

4. Model Definition

The function create\_densenet\_transfer\_learning\_model defines a DenseNet-based transfer learning model:

- The DenseNet201 pre-trained model is used as the base, loaded without the top classification layer.

- Its layers are frozen (not trainable) to retain the pre-trained features.

- Custom classification layers are added (GlobalAveragePooling2D, Dense, Dropout, and the final softmax output layer).

- The model is compiled with the Adam optimizer, a low learning rate, and categorical crossentropy loss for multi-class classification.

5. Training the Model

The model.fit function trains the model on the augmented training set:

- Uses callbacks like ReduceLROnPlateau (reduces learning rate when validation loss plateaus) and EarlyStopping (stops training early if validation accuracy doesn’t improve).

- Validates the model during training using the validation set.

6. Evaluation

The evaluate\_model function:

- Evaluates the model on the test set.

- Computes and prints metrics like test loss and accuracy.

- Predicts the test set classes and displays a detailed classification report.

7. Visualization

Various visualization functions:

- plot\_training\_history: Plots training and validation accuracy/loss over epochs.

- plot\_confusion\_matrix: Displays a heatmap of the confusion matrix for true vs. predicted classes.

- plot\_roc\_curve: Plots the Receiver Operating Characteristic (ROC) curve and calculates the Area Under Curve (AUC) for each class.

8. Overall Metrics

The evaluate\_and\_visualize\_metrics function combines evaluation steps:

- Computes the F1-score.

- Plots the confusion matrix and ROC curve.

- Provides a detailed analysis of model performance on the test set.

Key Takeaways

- Transfer Learning: DenseNet201 leverages pre-trained features, reducing the need for large datasets.

- Metrics: Accuracy, F1-score, confusion matrix, and ROC-AUC provide a comprehensive performance evaluation.

- Visualization: Plots help in understanding the training dynamics and evaluating model predictions.

**DenseNet**

1. Dataset Loading

The load\_flavia\_dataset\_from\_csv function reads a dataset from a CSV file, processes the image data, and prepares it for model training. It:

- Loads image paths and class labels.

- Resizes images to (299, 299) (the expected input size for most models like Xception or DenseNet).

- Normalizes image pixel values to [0, 1].

- One-hot encodes the labels for multi-class classification.

2. Data Splitting

The dataset is divided into:

- Training + validation set (80% of the total dataset).

- Test set (20% of the total dataset).

The training + validation set is further split into:

- Training set (80% of the 80% subset).

- Validation set (20% of the 80% subset).

3. Data Augmentation

The ImageDataGenerator applies transformations to training images to artificially expand the dataset:

- Rotation, width/height shifts, horizontal/vertical flips, zoom, and shear augmentations.

- These transformations make the model more robust to variations in the data.

4. Model Definition

The function create\_densenet\_transfer\_learning\_model defines a DenseNet-based transfer learning model:

- The DenseNet201 pre-trained model is used as the base, loaded without the top classification layer.

- Its layers are frozen (not trainable) to retain the pre-trained features.

- Custom classification layers are added (GlobalAveragePooling2D, Dense, Dropout, and the final softmax output layer).

- The model is compiled with the Adam optimizer, a low learning rate, and categorical crossentropy loss for multi-class classification.

5. Training the Model

The model.fit function trains the model on the augmented training set:

- Uses callbacks like ReduceLROnPlateau (reduces learning rate when validation loss plateaus) and EarlyStopping (stops training early if validation accuracy doesn’t improve).

- Validates the model during training using the validation set.

6. Evaluation

The evaluate\_model function:

- Evaluates the model on the test set.

- Computes and prints metrics like test loss and accuracy.

- Predicts the test set classes and displays a detailed classification report.

7. Visualization

Various visualization functions:

- plot\_training\_history: Plots training and validation accuracy/loss over epochs.

- plot\_confusion\_matrix: Displays a heatmap of the confusion matrix for true vs. predicted classes.

- plot\_roc\_curve: Plots the Receiver Operating Characteristic (ROC) curve and calculates the Area Under Curve (AUC) for each class.

8. Overall Metrics

The evaluate\_and\_visualize\_metrics function combines evaluation steps:

- Computes the F1-score.

- Plots the confusion matrix and ROC curve.

- Provides a detailed analysis of model performance on the test set.

Key Takeaways

- Transfer Learning: DenseNet201 leverages pre-trained features, reducing the need for large datasets.

- Metrics: Accuracy, F1-score, confusion matrix, and ROC-AUC provide a comprehensive performance evaluation.

- Visualization: Plots help in understanding the training dynamics and evaluating model predictions.

**DenseNet Implementation**

Custom DenseNet Architecture:

- DenseNet Base Model:

- Uses the DenseNet201 pre-trained on ImageNet as the base model.

- Removes the top classification layer (include\_top=False).

- Extracts hierarchical features through densely connected convolutional layers.

- Frozen Layers:

- All pre-trained layers are frozen to preserve ImageNet-trained features.

- Ensures these layers are not updated during training.

- Feature Extraction Layers:

- Includes Global Average Pooling (GAP) to reduce feature maps into a single vector per feature map.

- Adds fully connected layers (Dense layers) to learn task-specific patterns.

- Dense Layers:

- Adds a 1024-neuron layer with ReLU activation followed by a 512-neuron layer with ReLU activation.

- Incorporates Dropout layers (50% and 30%) to reduce overfitting.

- Output Layer:

- A softmax layer outputs probabilities for each class.

- Tailored to the number of categories in the new dataset.

---

**Compilation**:

- Optimizer:

- Adam optimizer with a learning rate of (10^{-4}) for smooth and gradual learning.

- Loss Function:

- Categorical cross-entropy to handle multi-class classification.

- Metrics:

- Tracks accuracy during training and validation.

---

**Xception Implementation**

Custom Xception Architecture:

- Xception Base Model:

- Leverages the Xception model pre-trained on ImageNet.

- Removes the top classification layer (include\_top=False).

- Captures rich features using depthwise separable convolutions for efficient computation.

- Frozen Layers:

- Retains pre-trained weights by freezing all base model layers.

- Prevents overwriting of general-purpose features.

- Feature Extraction Layers:

- Employs Global Average Pooling (GAP) to condense spatial feature maps into a single vector.

- Adds task-specific Dense layers for classification.

- Dense Layers:

- Includes a 1024-neuron Dense layer with ReLU activation followed by a 512-neuron Dense layer with ReLU activation.

- Uses Dropout layers to improve generalization (50% and 30% dropout rates).

- Output Layer:

- A softmax layer tailored to the dataset outputs class probabilities.

---

**Compilation**:

- Optimizer:

- Adam optimizer with a learning rate of (10^{-4}).

- Loss Function:

- Categorical cross-entropy for multi-class classification tasks.

- Metrics:

- Tracks accuracy to evaluate performance during training and validation.