Model Description

Datasets:

a. **Paddy Doctor: Paddy Disease Classification**

Link: https://www.kaggle.com/c/paddy-disease-classification/data

The training dataset consists of **10,407 (75%) labelled paddy leaf images across ten classes** (nine diseases and normal leaf). The dataset also contains additional metadata for each image, such as the paddy variety and age.

b. **Rice Leaf Disease Image Samples**

Link**:** https://data.mendeley.com/datasets/fwcj7stb8r/1

The data set contains **5,932 images** including four kinds of Rice leaf diseases i.e. **Bacterial blight, Blast, Brown Spot and Tungro.**

My approach – train both CNN and transformer model first with Metadata and then use these trained models to train another model for both CNN and transformer without metadata and do my final predictions using these models.

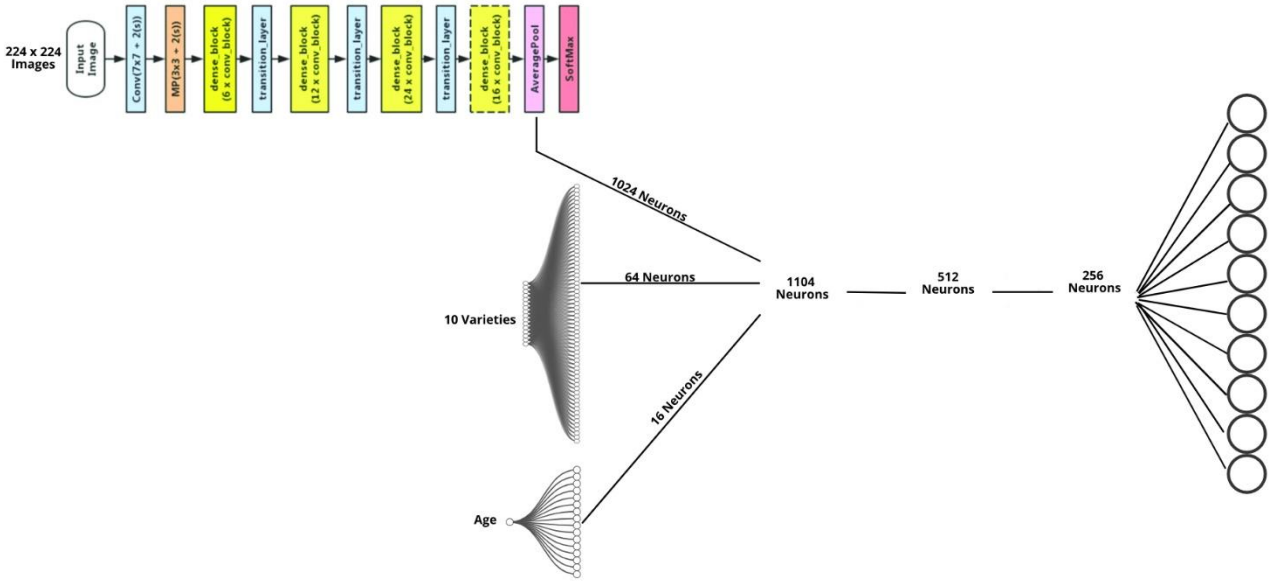
Part I) Classification using CNN

Training

Model 1: Images + Metadata

1. Pre-Processing:
    a. All Images should be uniform 224 x 224.
    b. All images should be RGB.
    c. Normalize all the Images
    d. Applied some random transformations to increase robustness (horizontal flip, rotation , affine colorjitter)
    e. convert into tensor
2. Attach Metadata – Now that we have our images , we need to attach metadata available, Wrote a function *PaddyLeafDatasetWithMetadata* to a attach images with metadata.
3. Do a 80-20 train-validation split
4. Create DataLoaders for training and validation set

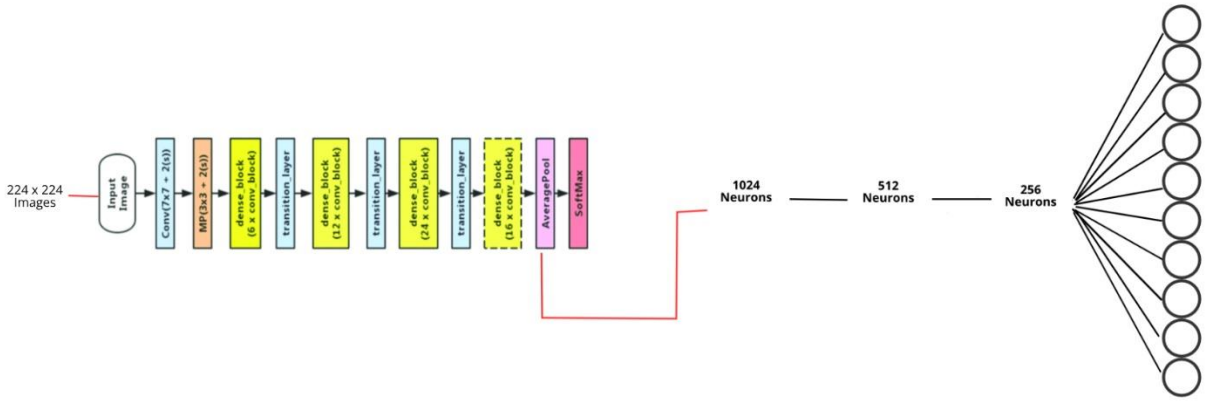
5. Define Model architecture :



   a. Used pretrained Densenet121
   b. Removed the last Fully connected layer of Denseney121 and took it's output after flattening which is a 1024-dim vector.
   c. We had 10 varieties , so I One-Hot Encoded them and passed it through a 10 → 64 Fully connected Layer.
   d. We also had age of plant which I passed through a 1→ 16 Fully Connected layer.
   e. Concatenated all these outputs (1024 + 64 + 16 = 1104).
   f. Passed it through a 1104 → 512 → 256 → 10 Fully Connected Multilayer Perceptron with ReLU activation and dropout = 0.3
   g. Finally Return the 10-dim output vector
6. Training the model:
   a. Optimizer = ADAM
   b. Loss Function = Cross Entropy Loss
   c. Learning Rate Scheduler = StepLR
   d. Trained for 10 Epochs
   e. Training Loop:
      i. Move every batch to Device (CUDA or CPU)
      ii. Calculate Loss
      iii. Backpropagate the Loss and update weights.
      iv. After every EPOCH calculate loss and accuracy on Validation set.
      v. Save optimizer state if current validation accuracy is better than previous best.
7. Evaluating Model :
   a. Wrote a function evaluate the model on validation set.
   b. It returns:
      i. Classification Report – Has accuracy , F1-score, Recall, Precision
      ii. Confusion Matrix – True vs Predicted Labels graph.
8. Plotting Results: Wrote 2 functions
   a. Plot_training_history – Plotted Accuracy vs Epoch and Loss vs Epoch for both training and validation.
   b. Plot_confusion_matrix – Plotted the confusion matrix obtained by validation set.

Model 2: Only Images

1. Define architecture:



   a. Now load the model created above
   b. Remove it's last 4 layers 1104,512,256,10
   c. Take the flattened output of remining model and push it through a 1024 → 512 → 256 →10 Fully Connected Multilayer Perceptron.
2. Load dataset 2 + dataset 2 which contains only images and apply the same transformations we applied while training model 1.
3. Train it using same optimizer , scheduler and loss function
4. You now Have a model which can predict class based solely on Image.

Note – The point of training model 1 and then fine tuning to make model 2 is that we have a large dataset in form of Dataset-1 with metadata , so a model trained on it will have high accuracy , then when we use this pre-trained model 1 to create a model 2 it achieves higher accuracy when only images are available as compared to training only model 2 without model 1.

Classification

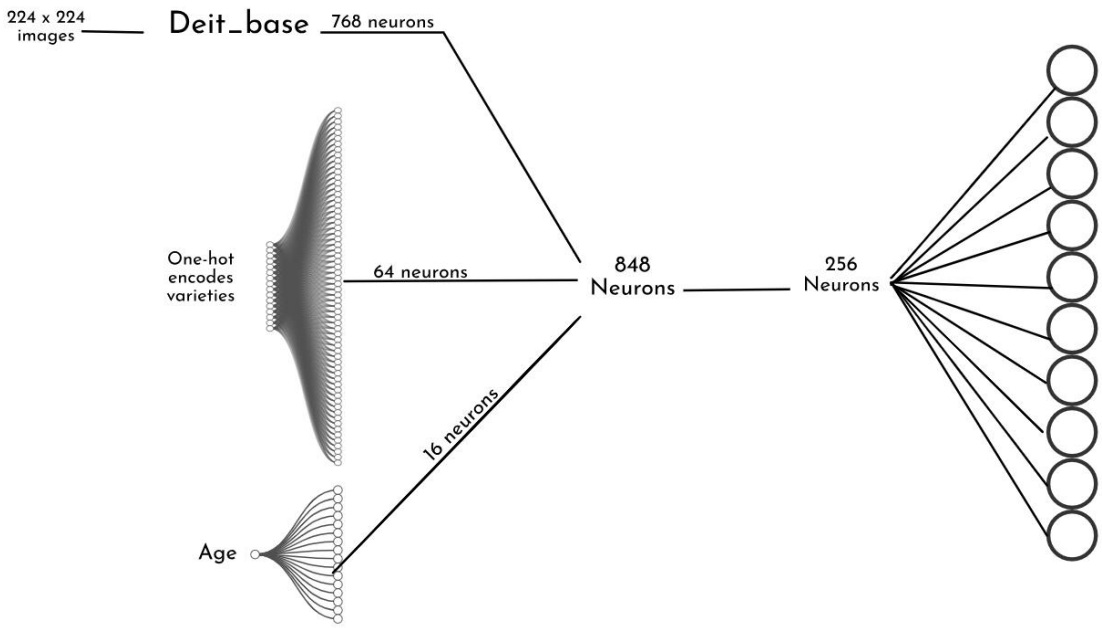
1. Pre-process test images:
   a. Make all images 224 x 224
   b. Make sure all images are RGB
   c. Normalize them
2. Attach metadata if available (we don't have any metadata in our case)
3. Prepare DataLoaders
4. Run these DataLoaders through Model 2 since we do not have metadata.
5. Save these results in a CSV names "CNN_classification.csv".

Part II) Classification using Transformers

Training

Model 1: Images + Metadata

1) Pre-Processing:
    a. All Images should be uniform 224 x 224.
    b. All images should be RGB.
    c. Normalize all the Images
    d. Applied some random transformations to increase robustness (horizontal flip, rotation , affine colorjitter)
    e. convert into tensor
2) Attach Metadata – Now that we have our images , we need to attach metadata available, Wrote a function *PaddyLeafDatasetWithMetadata* to a attach images with metadata.
3) Do a 80-20 train-validation split
4) Create DataLoaders for training and validation set
5) Define Model architecture :



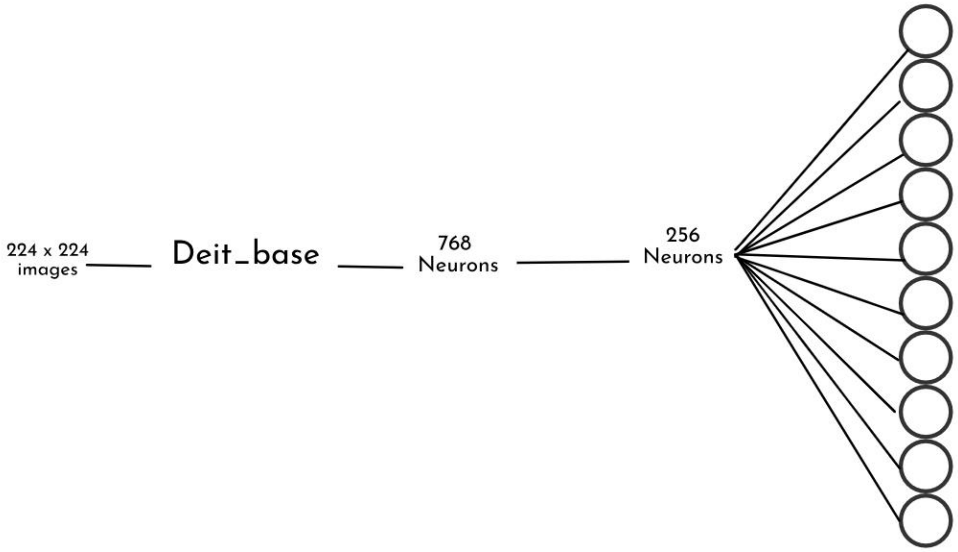
    a. Used pretrained Deit_base model
    b. Removed the classification head on Deit_base and took it's output which is a 768-dim vector.
    c. We had 10 varieties , so I One-Hot Encoded them and passed it through a 10 → 64 Fully connected Layer.
    d. We also had age of plant which I passed through a 1→ 16 Fully Connected layer.
    e. Concatenated all these outputs (768 + 64 + 16 = 848).
    f. Passed it through a 848 → 256 → 10 Fully Connected Multilayer Perceptron with ReLU activation and dropout = 0.3
    g. Finally Return the 10-dim output vector
6) Training the model:

a. Optimizer = ADAM
b. Loss Function = Cross Entropy Loss
c. Learning Rate Scheduler = StepLR
d. Trained for 20 Epochs
e. Training Loop:
    i. Move every batch to Device (CUDA or CPU)
    ii. Calculate Loss
    iii. Backpropagate the Loss and update weights.
    iv. After every EPOCH calculate loss and accuracy on Validation set.
    v. Save optimizer state if current validation accuracy is better than previous best.

7) Evaluating Model :
    a. Wrote a function evaluate the model on validation set.
    b. It returns:
        i. Classification Report – Has accuracy , F1-score, Recall, Precision
        ii. Confusion Matrix – True vs Predicted Labels graph.
8) Plotting Results: Wrote 2 functions
    a. Plot_training_history – Plotted Accuracy vs Epoch and Loss vs Epoch for both training and validation.
    b. Plot_confusion_matrix – Plotted the confusion matrix obtained by validation set.

Model 2: Only Images

1) Define architecture:



    a. Now load the model created above
    b. Remove it's last 4 layers 848,512,256,10
    c. Take the flattened output of remining model and push it through a 768 →256 → 10 Fully Connected Multilayer Perceptron.
2) Load dataset 2 + dataset 1 which contains only images and apply the same transformations we applied while training model 1.
3) Train it using same optimizer , scheduler and loss function
4) You now Have a model which can predict class based solely on Image.

Note – The point of training model 1 and then fine tuning to make model 2 is that we have a large dataset in form of Dataset-1 with metadata , so a model trained on it will have high accuracy , then when we use this pre-trained model 1 to create a model 2 it achieves higher accuracy when only images are available as compared to training only model 2 without model 1.

Classification

1) Pre-process test images:
    a. Make all images 224 x 224
    b. Make sure all images are RGB
    c. Normalize them
2) Attach metadata if available (we don't have any metadata in our case)
3) Prepare DataLoaders
4) Run these DataLoaders through Model 2 since we do not have metadata.
5) Save these results in a CSV names "Transformers_Classification.csv"

Hyperparameter Tuning

🔹 **Learning Rate: 0.001**

✓ Why?

- A learning rate of **0.001** is a **standard starting point** for fine-tuning pretrained models.
- It allows stable training without **diverging or overshooting** the optimal weights.
- For **CNNs (DenseNet-121)**, 0.001 works well for convergence.
- For **Transformers (DeiT-Base)**, lower learning rates (like 1e-4 or 1e-5) are sometimes better ,but **StepLR adjusts this dynamically**.

🔹 **Epochs: 10 for CNN, 20 for Transformer**

✓ Why?

- **CNNs (DenseNet-121):**
  - Since CNNs extract local patterns **efficiently**, fewer epochs (**10**) are enough to achieve strong performance.
  - Fine-tuning pretrained **DenseNet-121** typically **converges faster** than training from scratch.
- **Transformers (DeiT-Base):**
  - Transformers require **more epochs (20)** because they learn long-range dependencies and take time to adjust their feature representations.
  - Vision Transformers often need **more data and fine-tuning cycles** than CNNs.

🔹 **Batch Size: 32**

✓ Why?

- A batch size of **32** balances **stability and efficiency**.
- **Too small (e.g., 16):** Model updates are noisy and training is unstable.
- **Too large (e.g., 64+ on RTX 2050):** Can cause **memory issues** and slower convergence.

✓ Alternative Tuning Options:

- If **GPU memory allows**, try **batch size = 64** for better parallelization.
- If **VRAM is an issue**, reduce to **batch size = 16** and use **gradient accumulation**.

🔹 **Optimizer: Adam**

✓ Why?

- **Adam** (Adaptive Moment Estimation) dynamically adjusts the learning rate for each parameter, **helping with faster convergence**.
- It combines the benefits of **SGD with momentum** and **RMSProp**, making it **effective for deep learning models**.

🔹 **Learning Rate Scheduler: StepLR (Step Size = 7, Gamma = 0.1)**

✓ Why?

- **StepLR** reduces the learning rate by a factor of **0.1** every **7 epochs**, ensuring stable convergence.
- **For CNNs (10 epochs total):** Only **1 step down in LR** (at epoch 7).
- **For Transformers (20 epochs total):** Two reductions (at **epochs 7 & 14**) to refine learning.

🔹 **Loss Function: Cross Entropy**

✓ Why?

- **Cross Entropy Loss** is **ideal for multi-class classification** tasks.
- It measures how well the predicted class probabilities match the ground truth.

🔹 **Pretrained Models: DenseNet-121 (CNN) & DeiT-Base (Transformer)**

✓ Why DenseNet-121?

- **Compact (8M parameters)** but powerful due to **dense connections**.
- **Lower memory usage**, making it **efficient for fine-tuning** on 10K images.
- **Better feature reuse** than ResNet.

✓ Why DeiT-Base?

- **State-of-the-art Transformer for vision tasks**.
- DeiT is optimized for **data-efficient training**, making it **better suited for medium datasets (~10K images)**.
- **Stronger than ViT when trained on limited data**.

📌 **Some Recommendations**

- If someone a has lower compute power consider reducing batch size to 16, if someone has more compute then can try with batch size = 64
- AdamW optimizer can also be experimented with.
- Class weighting could be added to handle the slight disbalance in class-specific datasets

Comparative Report

1. Introduction

This report compares the performance of **Convolutional Neural Networks (CNNs)** and **Transformer-based models** for paddy leaf disease classification. The analysis considers classification accuracy, class-wise performance, the impact of metadata (variety and age), and training behaviour.

---

2. Model Performance Comparison

Model	Accuracy (With Metadata)	Accuracy (Without Metadata)
CNN	**73.17%**	**70.41%**
Transformer	**67.47%**	**73.53%**

Key Observations:

- **CNN performed better with metadata**, while its accuracy dropped slightly without metadata.
- **Transformer performed better without metadata**, but its accuracy significantly declined when metadata was introduced.
- **CNN benefited from metadata, whereas Transformer was negatively impacted by metadata.**

---

3. Class-wise Performance Analysis (used F1-Score)

Class	CNN (With Metadata)	CNN (Without Metadata)	Transformer (With Metadata)	Transformer (Without Metadata)
bacterial_leaf_blight	0.8514	0.8243	0.2899	0.8984
bacterial_leaf_streak	0.7012	0.6789	0.6846	0.6977
bacterial_panicle_blight	0.7234	0.6881	0.6080	0.7143
blast	0.6127	0.6914	0.6609	0.6411
brown_spot	0.5345	0.4792	0.5203	0.5076
dead_heart	0.8843	0.8721	0.8819	0.8807
downy_mildew	0.7382	0.6895	0.7004	0.7184
hispa	0.7512	0.7031	0.6656	0.7288

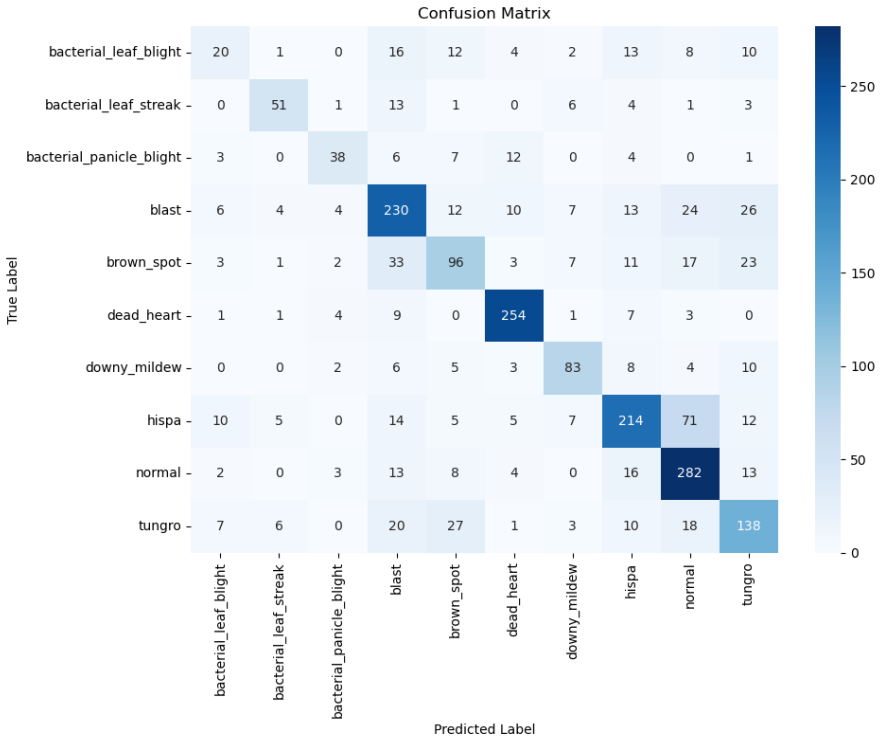
Class	CNN (With Metadata)	CNN (Without Metadata)	Transformer (With Metadata)	Transformer (Without Metadata)
normal	0.7214	0.7983	0.7334	0.7669
tungro	0.6532	0.7415	0.5923	0.8750

---

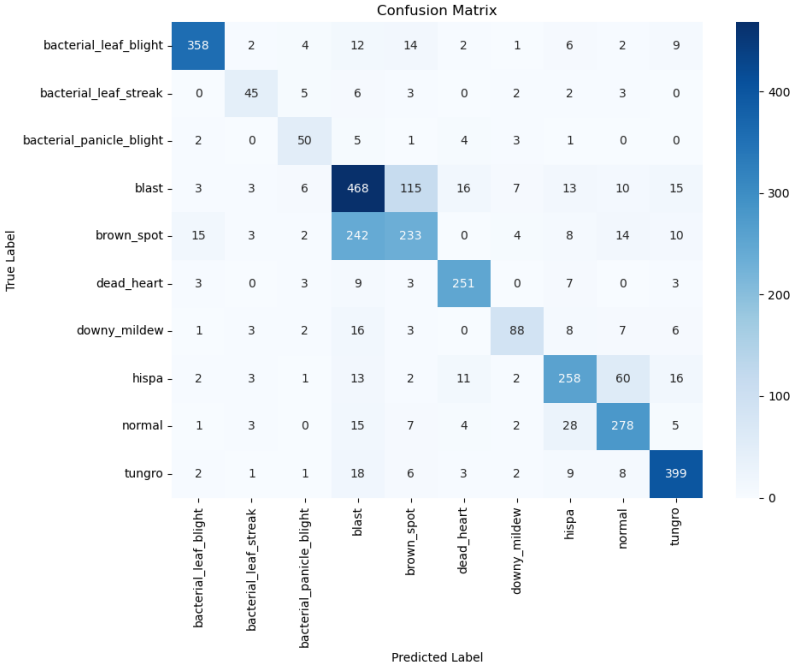
4. Confusion Matrix Analysis

To better understand the classification errors, below are the confusion matrices:

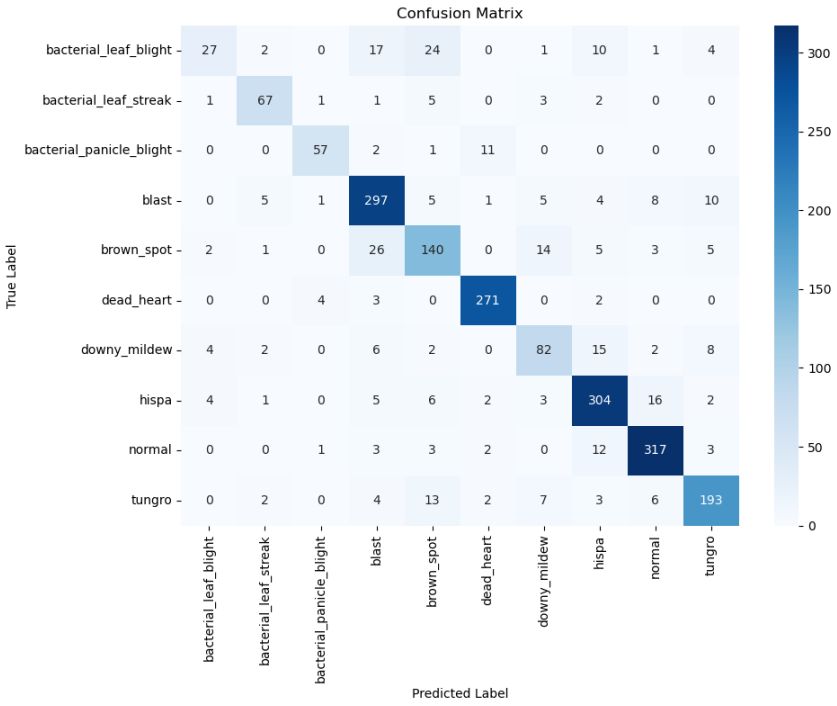
- **Transformer with Metadata**



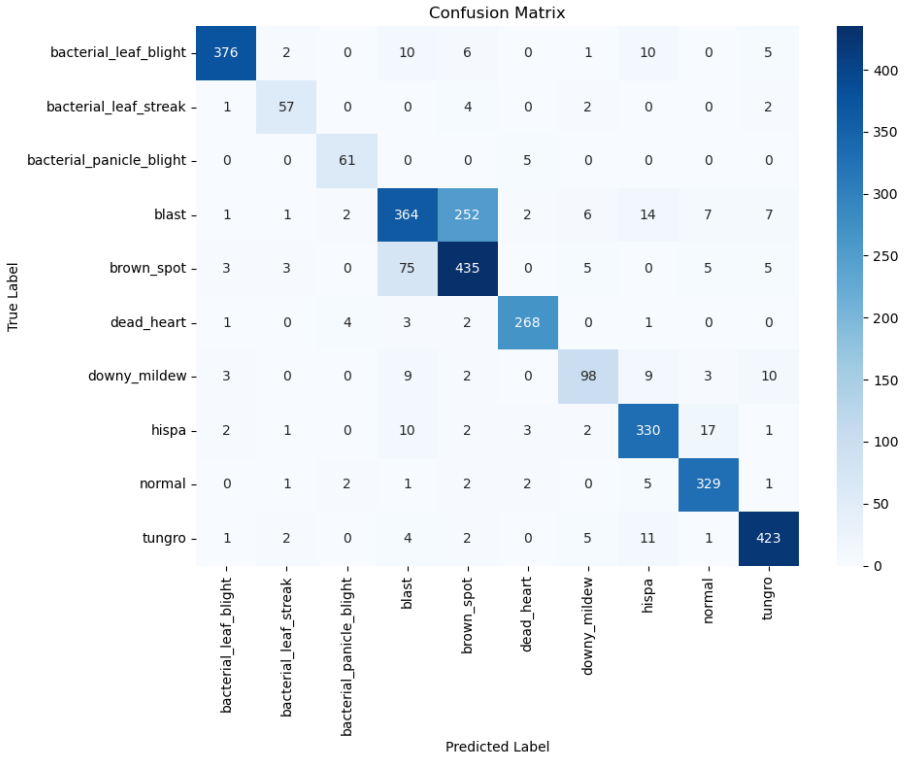
- **Transformer without Metadata**



- **CNN with Metadata**



- **CNN without Metadata**



Key Takeaways:

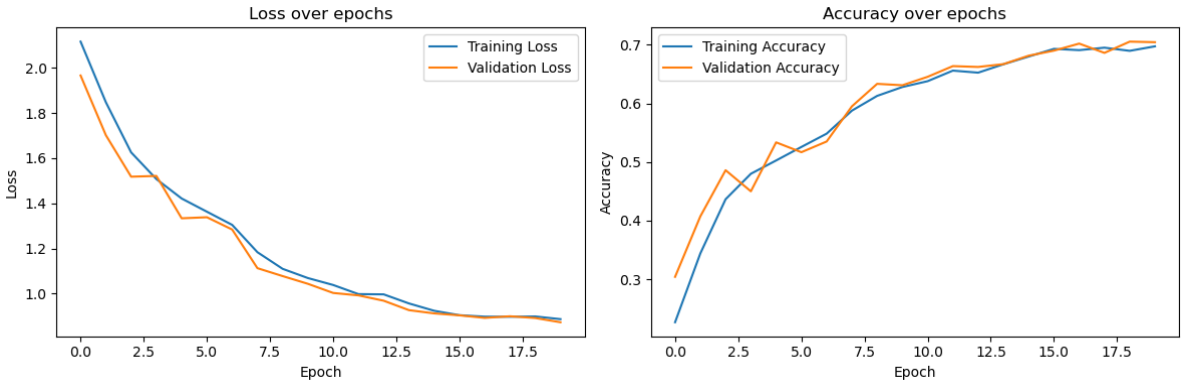
- **More misclassifications occurred when metadata was included in the Transformer model.**
- **On the contrary misclassifications were less when metadata was used**

---

5. Training Performance

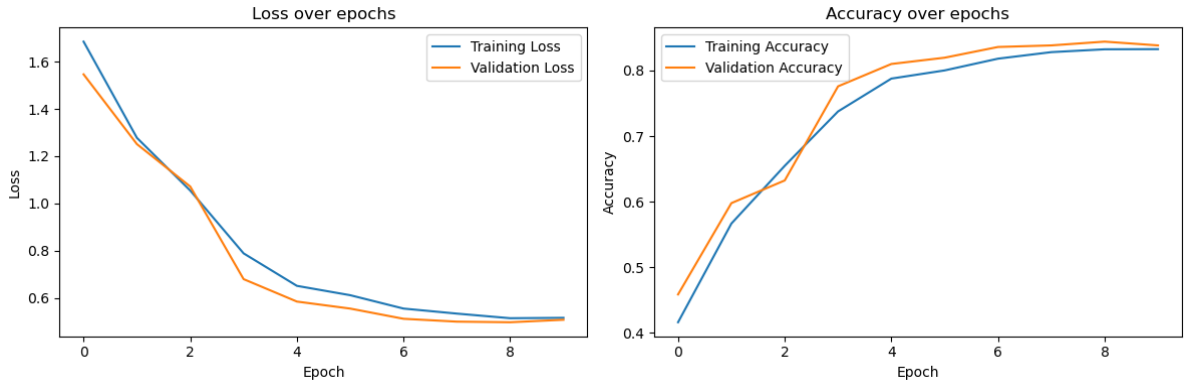
Training history graphs show how the models learned over time:

- **Transformer with Metadata**

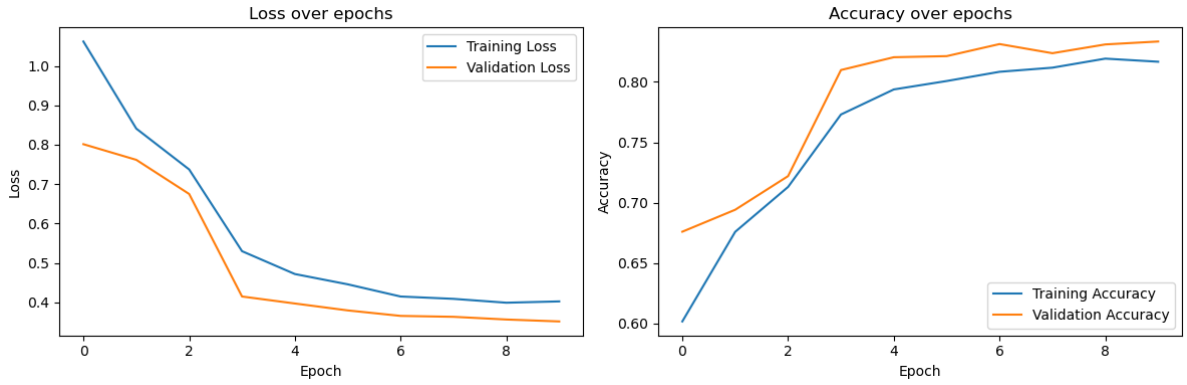


- **Transformer without Metadata**

- **CNN with Metadata**



- **CNN without Metadata**



Observations from Training Graphs:

- **With metadata, Transformer experienced slower convergence and fluctuating validation loss.**
- **Without metadata, the training curves showed better stability, leading to higher test accuracy.**
- **CNN models showed more stable training curves overall.**

---

6. Comparative Analysis

A. Metadata Impact

- **CNN benefited from metadata** (Accuracy ↑ **2.76%** with metadata).
- **Transformer suffered from metadata** (Accuracy ↓ **6.06%** with metadata).
- **CNN might be better at integrating structured metadata** due to its simpler architecture.
- **Transformer's late fusion technique may not effectively utilize metadata and could introduce noise.**

B. Generalization and Robustness

- **CNN had stable performance** across metadata conditions.
- **Transformer had more variance**, performing well without metadata but struggling with it.
- **CNN might be more robust** in real-world settings where metadata is useful.

C. Class-wise Performance

- **Transformer was significantly better for bacterial_leaf_blight without metadata (F1-Score: 0.8984).**
- **CNN performed consistently well for most classes when metadata was available.**
- **For dead_heart, both models performed equally well.**
- **Transformer was better for tungro disease without metadata.**

---

7. Advantages and Disadvantages

Feature	CNN	Transformer
Accuracy (with metadata)	**Higher (73.17%)**	Lower (67.47%)
Accuracy (without metadata)	Lower (70.41%)	**Higher (73.53%)**
Metadata Impact	**Positive (helps performance)**	**Negative (reduces performance)**
Computational Cost	Lower (faster training)	Higher (needs large datasets)
Generalization	Robust with metadata	Unstable with metadata
Class-wise Performance	More consistent across classes	Stronger in some classes, weaker in others
Overfitting Risk	Lower	Higher without careful tuning

---

8. Conclusion and Recommendations

- **CNN is recommended if metadata is useful, as it integrates structured metadata better.**

- **Transformer is preferable when metadata is unavailable or unreliable.**
- **For real-world deployment, CNN is more stable and computationally efficient.**
- **Future work should investigate better metadata integration for Transformers (e.g., early fusion, better embeddings).**

Final Recommendation

Model Recommendation for Paddy Leaf Disease Classification

1. Summary of Findings

- **CNN performs better with metadata (73.17%)**, while **Transformer excels without metadata (73.53%)**.
- **Transformers struggle to utilize metadata effectively**, leading to lower accuracy when included.
- **CNN offers more stable performance and is computationally efficient**, making it preferable for deployment.

2. Model Selection Guide

Use Case	Recommended Model	Reason
With metadata	**CNN**	Higher accuracy, stable performance
Without metadata	**Transformer**	Performs better without metadata
Limited computing power	**CNN**	Lower resource requirements
Field deployment	**CNN**	Reliable, consistent predictions
Targeting specific diseases	**Hybrid/Ensemble**	Custom tuning for best results

3. Future Recommendations

1. **Enhance metadata integration in Transformers** using better encoding methods.
2. **Explore hybrid models (CNN + Transformer)** to leverage both strengths.

4. Conclusion

CNNs are the **preferred choice for deployment**, while Transformers **need better metadata fusion** or **larger training dataset** for reliability. Hybrid approaches may offer the best of both.