Vietnam General Confederation of Labor
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY

KNOWLEDGE DISCOVERY AND DATA MINING

Instructor: **Mr. HOANG ANH**

Student: **PHAM LE QUOC DAT - 521K0128**

DUONG NGOC BAO NHI - 521K0143

Year **: 2023-2024**

HO CHI MINH CITY, 2024

Vietnam General Confederation of Labor
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY

KNOWLEDGE DISCOVERY AND DATA MINING

Instructor: **Mr. HOANG ANH**

Student: **PHAM LE QUOC DAT - 521K0128**

DUONG NGOC BAO NHI - 521K0143

Year **: 2023-2024**

HO CHI MINH CITY, 2024

ACKNOWLEDGEMENT

We, as a team, wish to express our sincere gratitude and deep appreciation to Mr. Hoang Anh for his invaluable guidance and support throughout our final project. His profound expertise and insightful mentorship played an indispensable role in facilitating our progress and driving our project forward. Mr. Anh's profound knowledge and understanding of advanced data mining techniques and methodologies enabled us to overcome intricate challenges, expand our conceptual understanding, and approach the final project with renewed confidence and enthusiasm. We are truly grateful for his unwavering dedication, which not only deepened our mastery of data mining concepts but also inspired us to strive for excellence and give our utmost efforts. His guidance was instrumental in our learning journey and the successful culmination of this final project endeavor.

Ho Chi Minh city,20th May , 2024

Author

(Sign and write full name)

Pham Le Quoc Dat

Duong Ngoc Bao Nhi

CONFIRMATION AND ASSESSMENT SECTION

Instructor confirmation section

_____

_____

_____

_____

_____

_____

_____

Ho Chi Minh , 2024

(Sign and write full name)

Evaluation section for grading instructor

_____

_____

_____

_____

_____

_____

_____

Ho Chi Minh, 2024

(Sign and write full name)

THE PROJECT IS COMPLETED
AT TON DUC THANG UNIVERSITY

Our team would like to assure that this is our own research project and is under the scientific guidance of Hoang Anh. The research content and results in this topic are honest and have not been published in any form before. The data in the tables for analysis, comments, and evaluation were collected by the author from different sources and clearly stated in the reference section.

In addition, the report also uses a number of comments, assessments as well as data from other authors and other organizations, all with citations and source notes.

If any fraud is detected, our team will take full responsibility for the content of our IT Project Report. Ton Duc Thang University is not involved in copyright violations caused by us during the implementation process (if any).

Ho Chi Minh city, , 2024

Author

(Sign and write full name)

Pham Le Quoc Dat

Duong Ngoc Bao Nhi

INDEX

Introduction

Ensuring corn kernel quality is crucial in the agricultural industry, but manual inspection is time-consuming and error-prone. This code explores the use of deep learning techniques, specifically ResNet50 with a Luong Attention mechanism and Vision Transformer (ViT), for the task of automated corn kernel quality classification.

The code implements a comprehensive pipeline for data preparation, model definition, training, and evaluation of these two architectures. ResNet50, a popular convolutional neural network (CNN), is combined with attention to focus on relevant image features, while ViT leverages self-attention to capture global relationships within images.

By comparing the performance of these models through metrics like accuracy and confusion matrices, the code aims to provide insights into the effectiveness of CNNs and ViTs for corn kernel quality classification. The results can inform the selection of appropriate deep learning models for similar applications in the agricultural domain, potentially leading to more efficient and accurate quality assessment processes.

Classify Corn images using Convolutional Neural Network and Vision Transformer

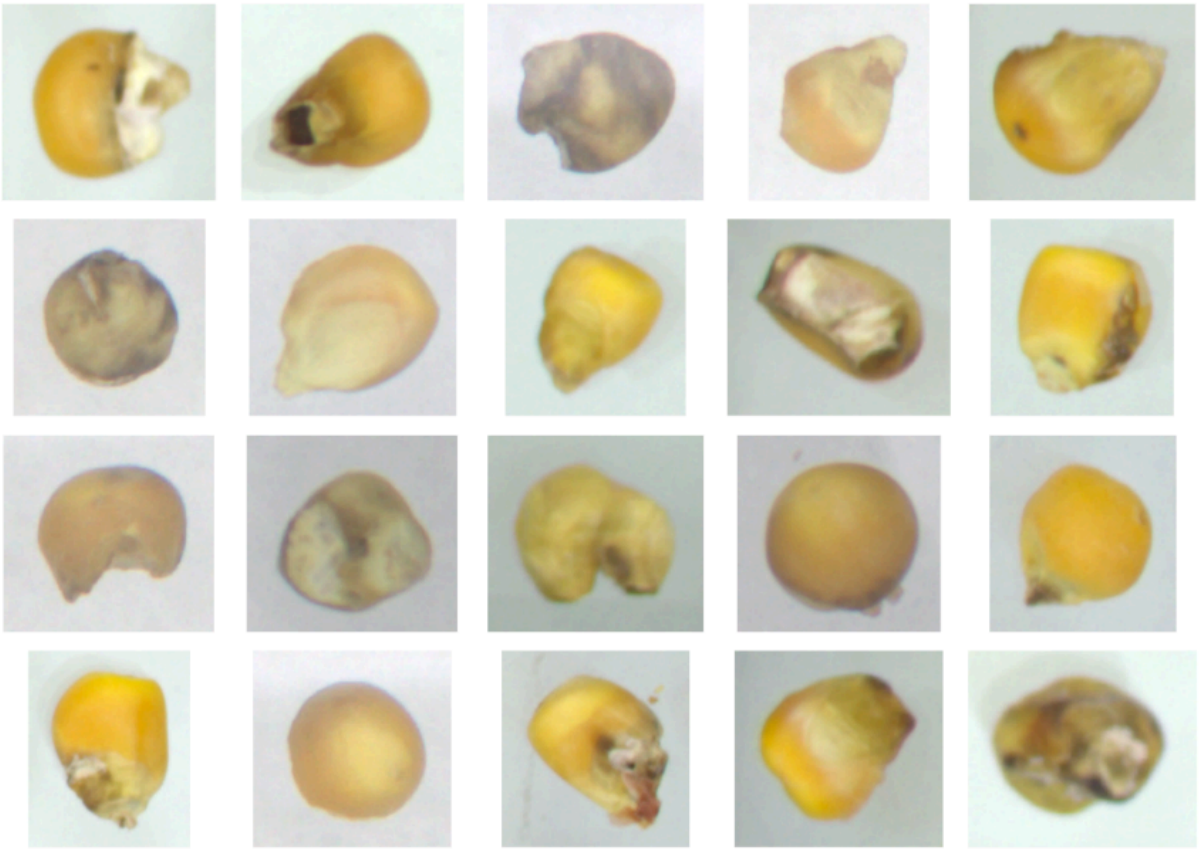

Figure 1: A random sample from the data set

Problem

The goal of this project is to develop a machine learning model capable of categorizing corn kernel images into four distinct classes: Pure, Broken, Discolored, and Silkcut. Manual inspection is not scalable due to the need for human resources, inconsistency between inspectors, and a slower processing pipeline.

Fine-grained objects (like seeds) are visually similar at a glance, and details in discriminative local regions are required for correct recognition.

Images are taken for both sides of the corn, top-view (8901) and bottom-view (8901). The primary dataset is highly imbalanced.

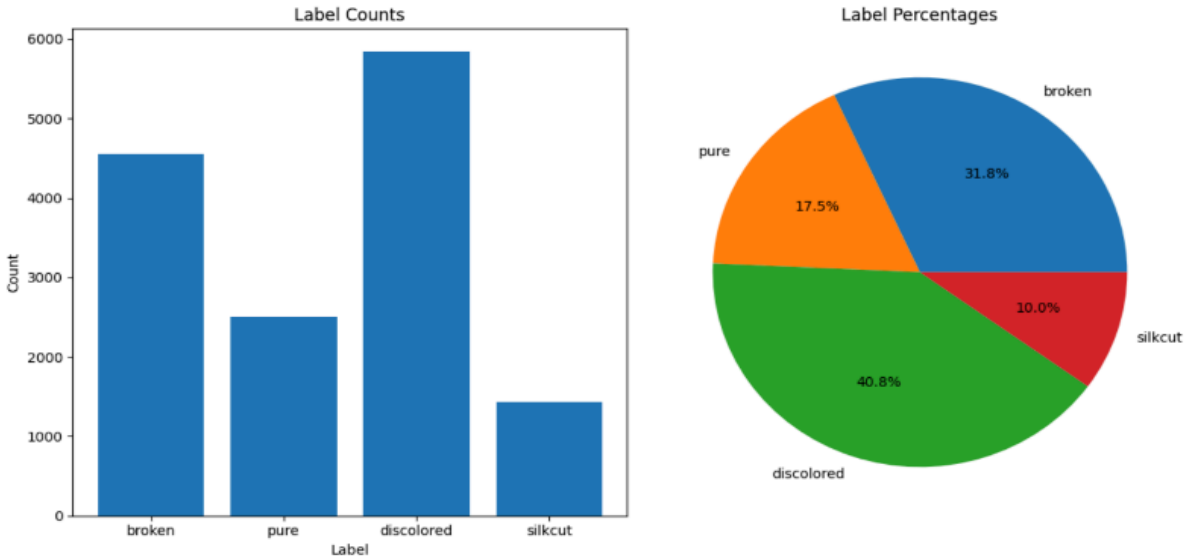

Figure 2: Distribution of class labels

The dataset consists of a total of 14,322 corn kernel images, each labeled into one of the four classes.

The class distribution is as follows:

- Pure: 5,837 images (17.5%)
- Broken: 4,554 images (31.8%)
- Discolored: 2,504 images (40.8%)
- Silkcut: 1,427 images (10.0%)

The most common class is Discolored (40.8%), followed by Broken (31.8%). Both Pure and Silkcut classes are underrepresented.

Each class represents a specific condition or defect that can significantly impact the quality and usability of the corn kernels.

- The Pure class comprises intact, undamaged kernels with a consistent color and appearance. These are typically considered high-quality.

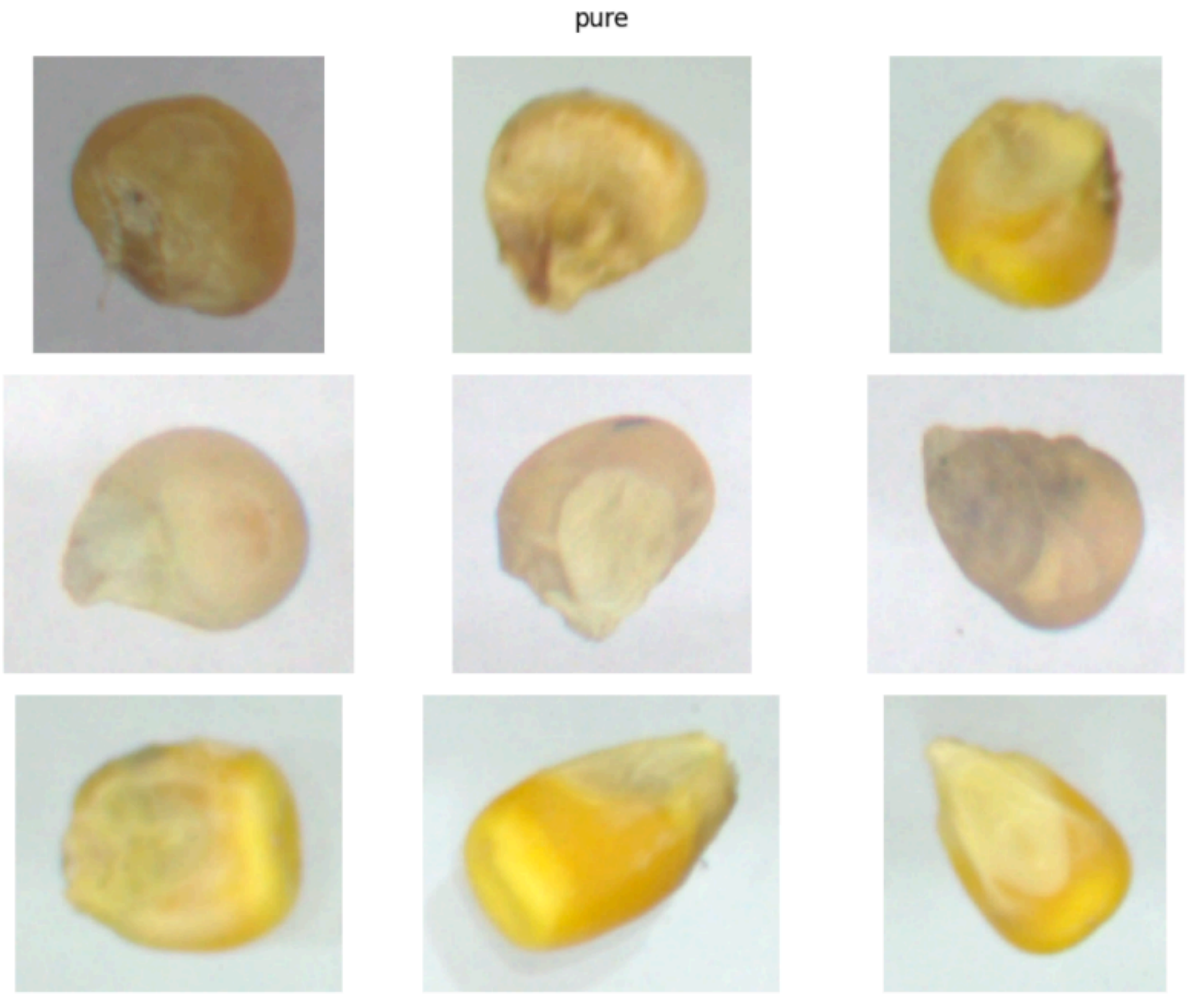

Figure 3: Random samples of Pure class

- The Broken class encompasses kernels that have been physically damaged or fractured, often due to mechanical stress during harvesting, transportation, or handling processes.

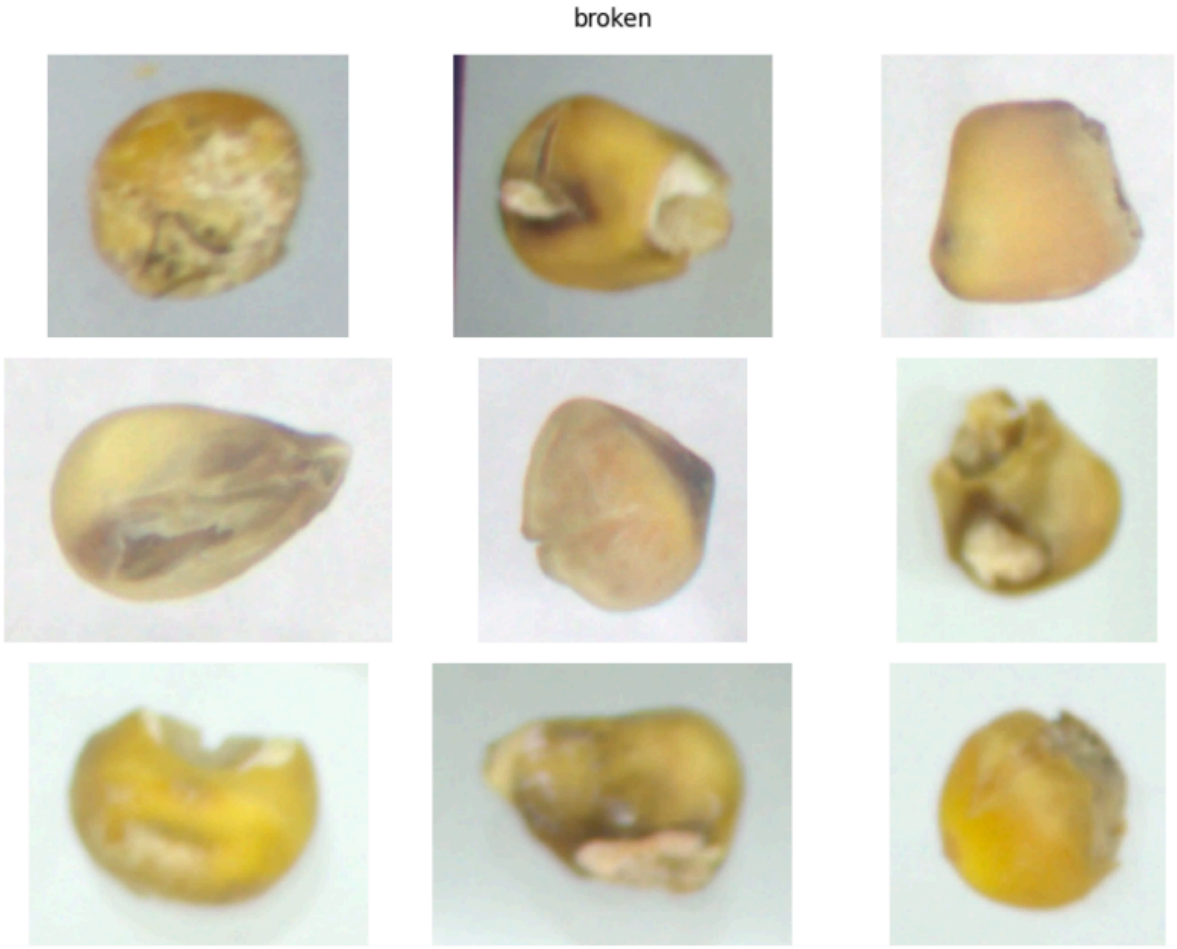

Figure 4: Random samples of Broken class

- The Discolored class includes kernels that have undergone color changes, either due to environmental factors or fungal/bacterial infections.

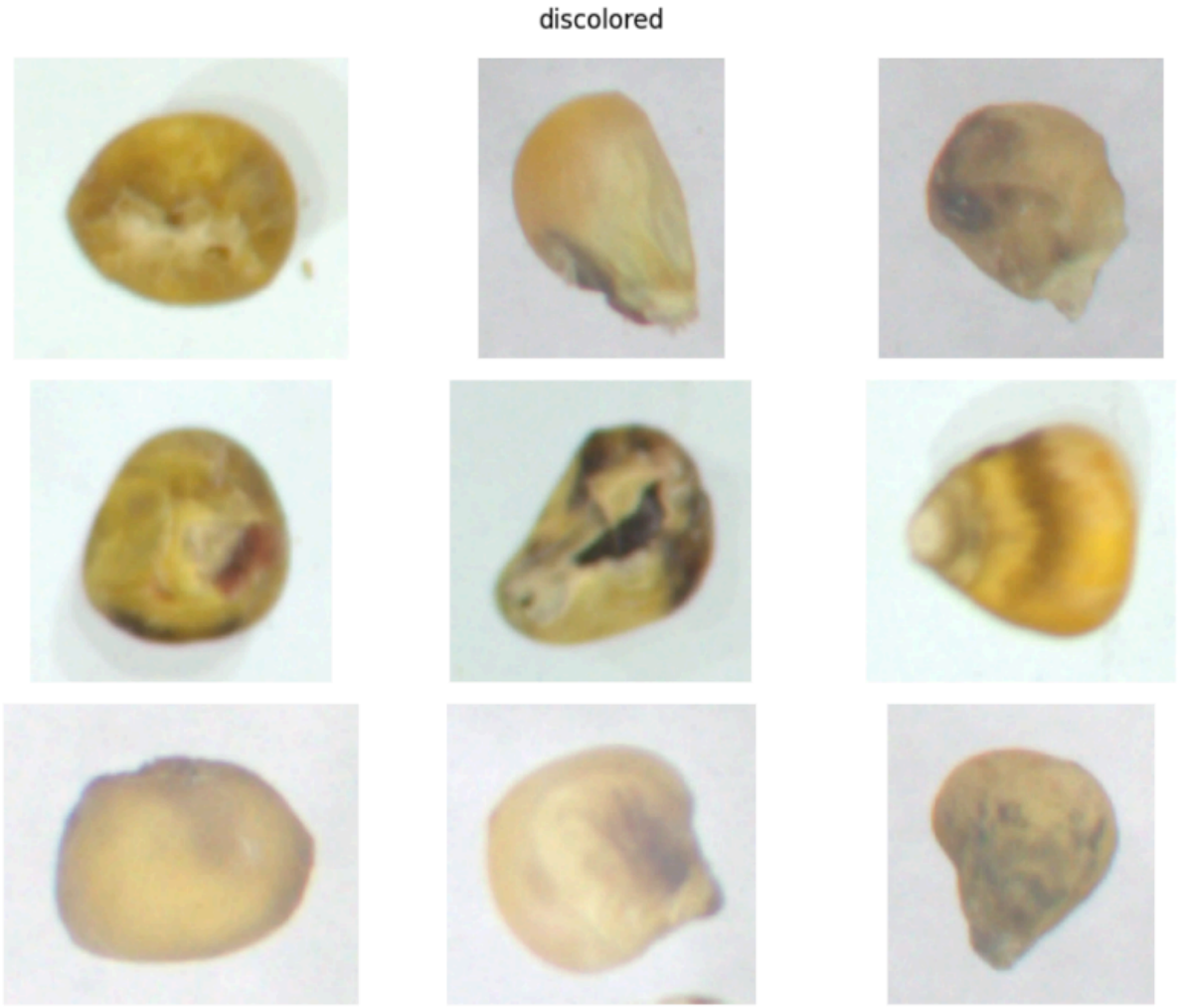

Figure 5: Random samples of Discolored class

- The Silkcut class refers to kernels that have been damaged or cut by the silk strands during the pollination process.

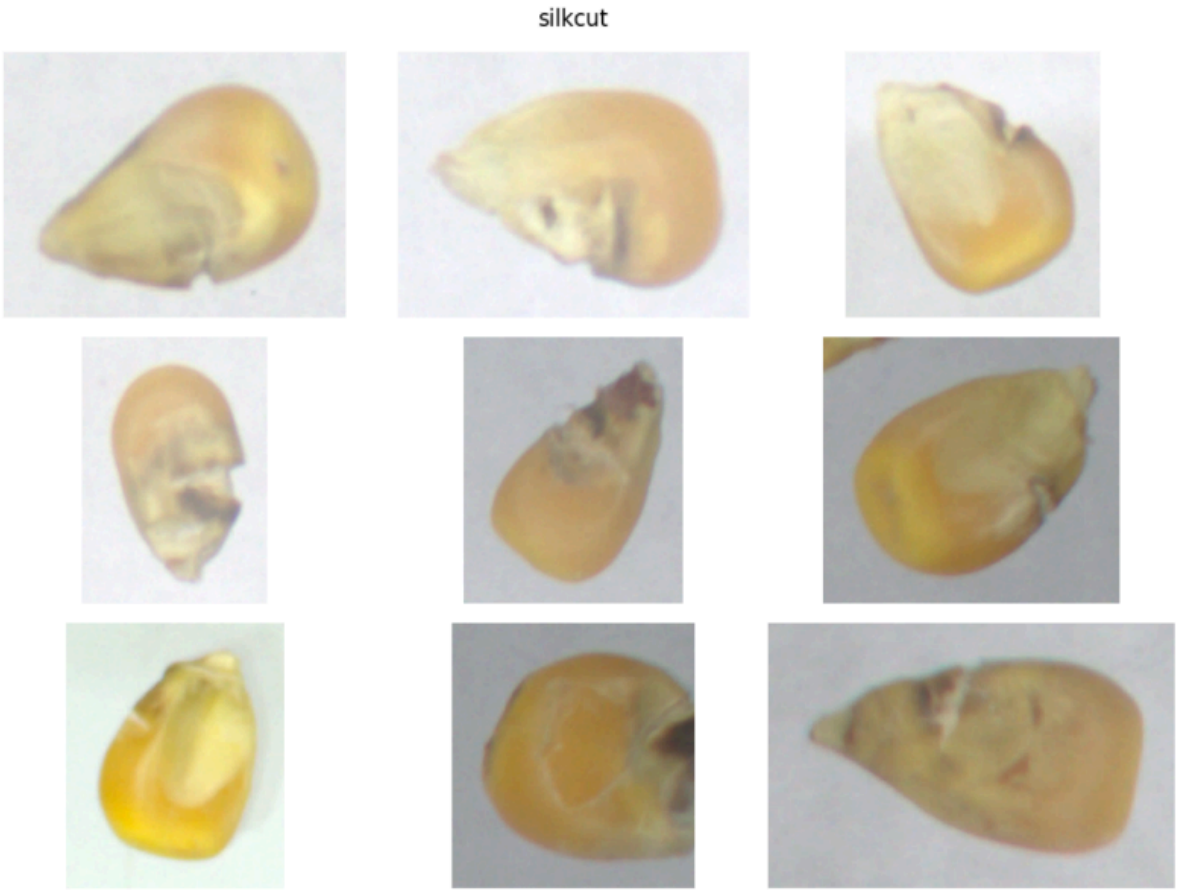

Figure 6: Random samples of Silkcut class

Approach

Data Processing

The dataset is structured as a CSV file with the following columns:

1. seed_id: A unique identifier for each corn kernel sample.
2. view: Indicates the perspective from which the image was taken, either "top" or "bottom".
3. image: The file path of the corresponding corn kernel image.
4. label: The class label, which can be "pure", "broken", "discolored", or "silkcut".

For classification, only the Image was used as the input. View was not used.

	seed_id	view	image	label
0	0	top	train/00000.png	broken
1	1	bottom	train/00001.png	pure
2	3	top	train/00003.png	broken
3	4	top	train/00004.png	pure
4	5	top	train/00005.png	discolored
...	...	...	...	...
14317	17795	top	train/17795.png	pure
14318	17796	top	train/17796.png	discolored
14319	17797	top	train/17797.png	broken
14320	17799	bottom	train/17799.png	pure
14321	17800	bottom	train/17800.png	discolored

Figure 7: First 5 and last 5 rows of the data set

The images are provided in Portable Network Graphics (PNG) format and are located in the "train" directory. Each image file is named using the seed_id, for example, "train/00000.png".

To prepare the data for training and evaluation, several preprocessing steps were applied:

1. Resizing: All images were resized to a fixed size of 224 x 224 pixels to ensure consistent input dimensions for the model.

2. Normalization: The pixel values of the images were normalized using mean subtraction and standard deviation scaling. Specifically, the means (0.485, 0.456, 0.406) and standard deviations (0.229, 0.224, 0.225) were calculated over the ImageNet dataset and used for normalization.

3. Data Augmentation: To increase the diversity of the training data and improve model generalization, various data augmentation techniques were applied. These included:
    - Random rotation (with a probability of 0.5 and border replication)
    - Normalization (using the aforementioned means and standard deviations)
    - Conversion to PyTorch tensor format

The data augmentation transformations were applied on-the-fly during training using the Albumentations library. For the validation set, only resizing and normalization were applied without any random augmentations
The transformations were defined as follows:

```
valid_tfms = A.Compose([
    A.Resize(width=224, height=224),
    A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])

train_tfms = A.Compose([
    A.Resize(width=224, height=224),
    A.Rotate(p=0.5, border_mode=cv2.BORDER_REPLICATE),
    A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])
```

Models

Traditional fully-connected neural networks (FFNNs) or other classical machine learning models are not well-suited for image data, as images are high-dimensional matrices. Feature extraction/feature engineering is required to effectively process image data by:

1. Extracting relevant features from the image
2. Performing classification using a neural network on the extracted features

The proposed solution involves:

- Using a pre-trained ResNet50 as a feature extractor
- Applying the Luong Attention mechanism for global feature attention
- Classifying using the attended features
- Fine-tuning the ResNet50 in later epochs

We also experimented with Vision Transformer (ViT), a transformer-based architecture for image classification. The proposed ViT solution is similar:

- Using a pre-trained ViT as a feature extractor
- Classifying using the Class [CLS] token

With ViT, we did not use an additional attention mechanism, as the model already contained self-attention mechanisms. Since the ViT model is large, fine-tuning it on consumer GPUs is difficult. Therefore, we only trained a fully-connected neural network (FFNN) for classification on the extracted Class [CLS] token.

Convolutional Neural Network

Convolutional Neural Networks (CNNs) are a powerful class of neural networks designed to process data with a grid-like topology, such as images. CNNs are comprised of three types of layers: convolutional layers, pooling layers, and fully-connected layers.

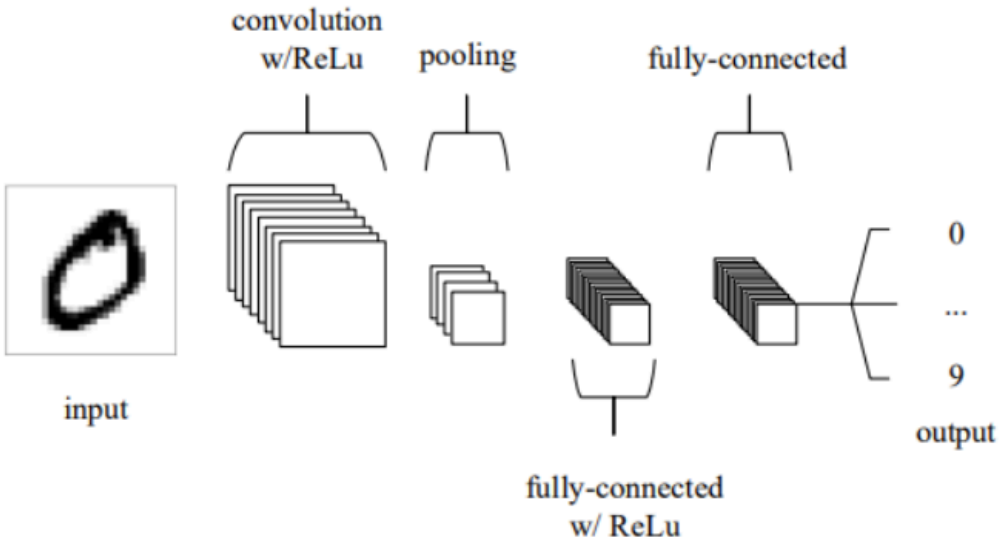

Figure 8: An simple CNN architecture, comprised of just five layers

The basic functionality of the example CNN above can be broken down into four key areas:

1. The input layer holds the pixel values of the image.

2. The convolutional layer determines the output of neurons connected to local regions of the input through the calculation of the scalar product between their weights and the region connected to the input volume. The rectified linear unit (ReLU) activation function is applied.

3. The pooling layer performs downsampling along the spatial dimensionality of the given input, further reducing the number of parameters within that activation.

4. The fully-connected layers perform the same duties as standard artificial neural networks(ANNs) and attempt to produce class scores from the activations for classification or regression.

Through this method of transformation, CNNs transform the original input layer by layer using convolutional and downsampling techniques to produce class scores. Convolutional layer parameters focus on the use of learnable kernels. These kernels are small in spatial dimensionality but spread along the depth of the input. When the data hits a convolutional layer, the layer convolves each filter across the spatial dimensionality of the input to produce a 2D activation map.

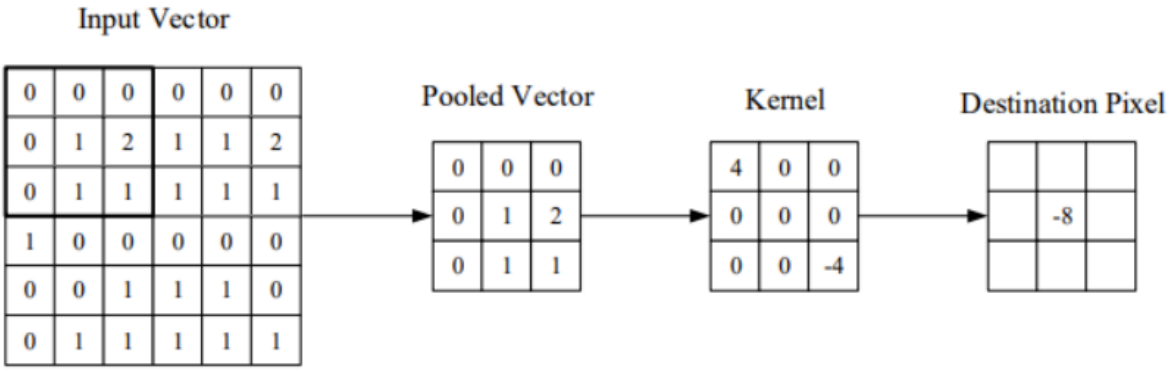

Figure 9: A visual representation of a convolutional layer. The center element of the kernel is placed over the input vector, and a weighted sum of itself and nearby pixels is calculated.

As the kernel slides across the input, the scalar product is calculated for each value in the kernel. The network learns kernels that "fire" when they see specific features at a given spatial position of the input. These are commonly known as activations. Each

kernel has a corresponding activation map, and these are stacked along the depth dimension to form the full output volume from the convolutional layer.
There are different types of pooling, with the two most popular being:

1. Max Pooling

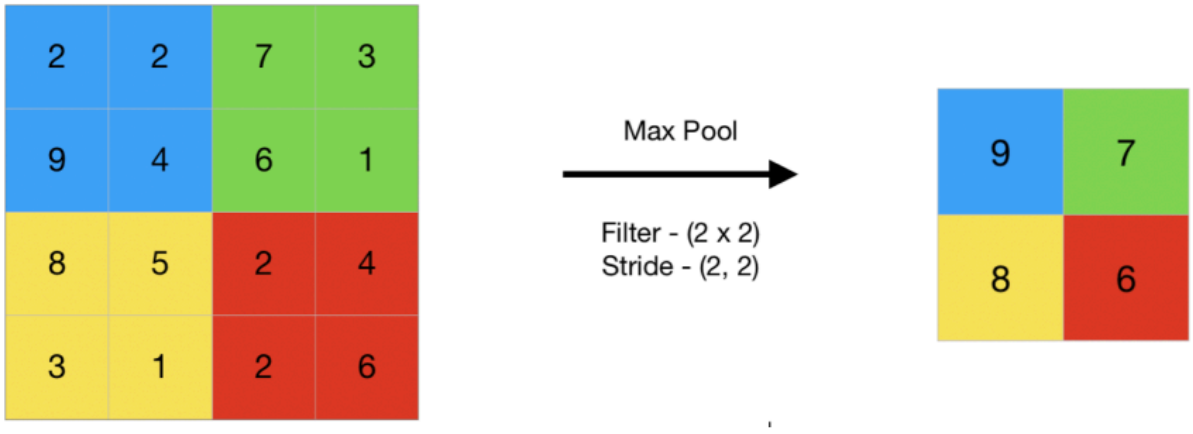

Figure 10: Max Pooling

Max pooling selects the maximum element from the region of the feature map covered by the filter. Thus, the output after the max-pooling layer would be a feature map containing the most prominent features of the previous feature map.

2. Average Pooling

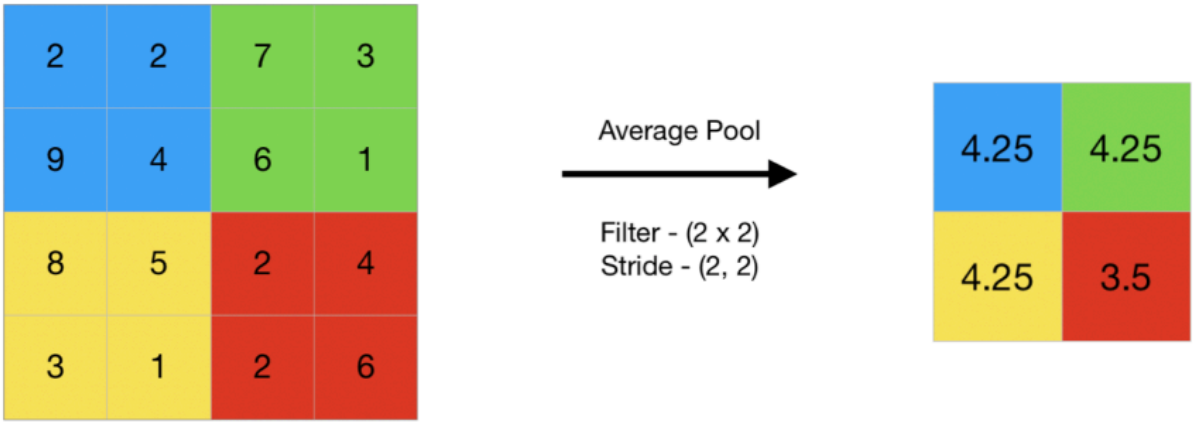

Figure 11: Average Pooling

Average pooling computes the average of the elements present in the region of the feature map covered by the filter. While max pooling gives the most prominent feature in a particular patch of the feature map, average pooling gives the average of features present in a patch.

ResNet50

The CNN architecture used in this model is based on the ResNet50 architecture, pre-trained on the ImageNet dataset, and serves as a powerful feature extractor.

Deep convolutional neural networks naturally integrate low/mid/high-level features and classifiers in an end-to-end multilayer fashion, and the "levels" of features can be enriched by increasing the number of stacked layers (depth).

However, an obstacle was the notorious problem of vanishing/exploding gradients, which hamper convergence from the beginning.

The exploding gradient problem is understood as follows:

> When the error is back propagated through a neural network, it may increase exponentially from layer to layer. In those cases, the gradient with respect to the parameters in lower layers may be exponentially greater than the gradient with respect to parameters in higher layers. This makes the network hard to train if it is sufficiently deep.

Similarly, the vanishing gradient problem occurs when the gradient decreases exponentially as it goes through deeper layers.

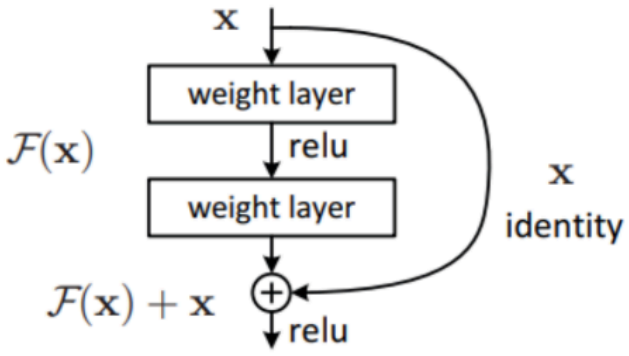

Figure 12: Residual learning: a building block

To mitigate this problem, residual learning is proposed, where the unmodified input (identity) is added to the output of the layer.

Identity shortcut connections add neither extra parameters nor computational complexity.

A residual connection in a layer means that the output of a layer is a convolution of its input plus its input.

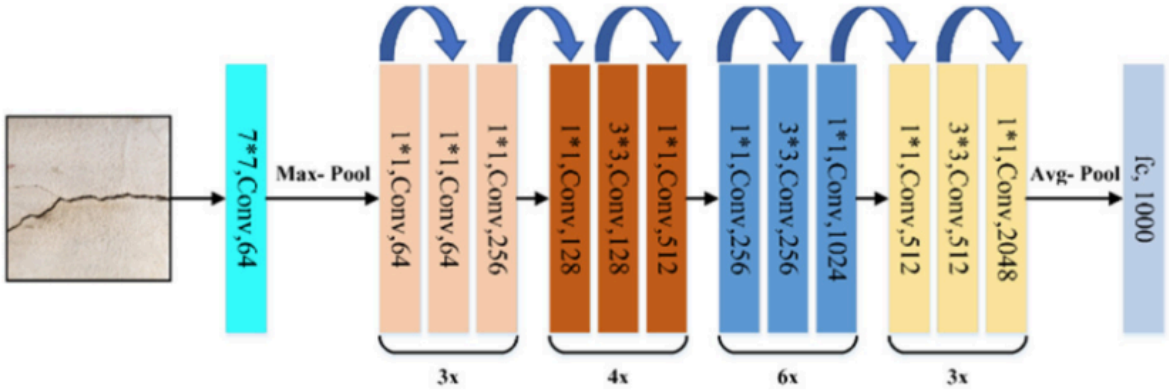

Figure 13: The architecture of ResNet-50 model

For the current application:

- The ResNet-50 model is truncated to remove the final fully connected layer, keeping only the convolutional layers.
- During the forward pass, the input images are processed through the ResNet50 convolutional layers, and the output features are reshaped and permuted to match the expected input format for the attention mechanism.
- The output from these convolutional layers is reshaped to form a feature map of shape (batch_size, 196, 1024), where 196 is the spatial dimension (14x14) and 1024 is the depth of the feature map.

```
class EncoderCNN(nn.Module):
    def __init__(self):
        super(EncoderCNN, self).__init__()
        resnet =
models.resnet50(weights=models.ResNet50_Weights.IMAGENET1K_V2)

        modules = list(resnet.children())[:-3]

        self.resnet = nn.Sequential(*modules)
```

```
    def forward(self, images):
        features = self.resnet(images)
        batch_size = images.size(0)
        features = features.reshape(batch_size, 1024, -1).permute(0, 2,
1)

        return features
```

ImageNet Data set

The ResNet50 and Vision Transformer (ViT) models used in this project were pre-trained on the ImageNet dataset, a large-scale database of annotated images organized according to the WordNet hierarchy, containing over 14 million images spanning 21,841 synsets (sets of synonyms).

Pre-training on ImageNet enables the models to learn rich, transferable feature representations that can be fine-tuned for specific tasks like corn kernel classification. The diverse range of object categories, viewpoints, backgrounds, and lighting conditions in ImageNet helps develop robust and generalizable models.

During pre-training, the models learn high-level features capturing semantic content, as well as low-level features capturing textures, edges, and visual patterns by training on the ImageNet classification task of predicting the correct object category.

Leveraging this pre-trained knowledge allows the ResNet50 and ViT models to be effectively fine-tuned on the corn kernel dataset, requiring fewer training examples and computational resources compared to training from scratch.

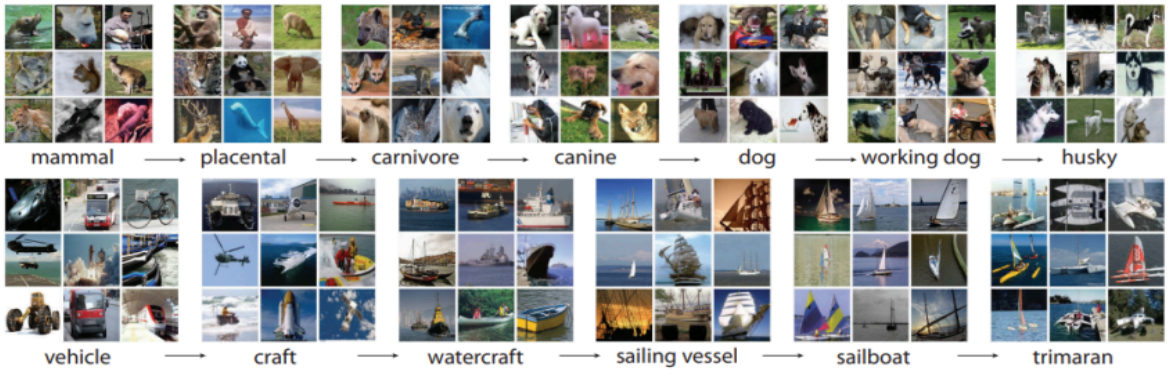

Figure 14: A snapshot of two root-to-leaf branches of ImageNet: the top row is from the mammal subtree; the bottom row is from the vehicle subtree. For each synset, 9 randomly sampled images are presented.

Luong Attention

The Luong Attention mechanism, introduced in the paper "Effective Approaches to Attention Based Neural Machine Translation", is used to attend to the most relevant features extracted by the CNN. This attention mechanism helps the model focus on the important regions of the input images, potentially improving the classification performance.

Luong Attention computes a context vector based on the alignment scores between the query (same as the key in this case) and the keys (the encoder output features).

The LuongAttention module implements the Luong Attention mechanism:

```
class LuongAttention(nn.Module):
    def __init__(self, hidden_size):
        self.hidden_size = hidden_size

    def forward(self, query, keys):
        scores = torch.bmm(query, keys.transpose(1, 2))
        scores = scores / math.sqrt(self.hidden_size)
        weights = F.softmax(scores, dim=-1)
        context = torch.bmm(weights, keys)

        return context, weights
```

The model

- The LuongAttention module takes the features from the EncoderCNN as input and computes attention scores based on the similarity between the query (current features) and the keys (all features). These attention scores are then used to weight the features and obtain a weighted combination of the features, called the context vector.

- The ClassificationHead module is responsible for taking the attended features and performing the final classification into the four classes (Pure, Broken, Discolored, Silkcut):

```
class ClassificationHead(nn.Module):
  def __init__(self, encoder_dim, num_classes):
    super(ClassificationHead, self).__init__()
    self.fc = nn.Linear(encoder_dim * 196, num_classes)
    self.dropout = nn.Dropout(0.5)
    self.bn = nn.BatchNorm1d(num_classes)

  def forward(self, features):
    batch_size, _, _ = features.shape
    features = features.reshape(batch_size, -1)
    scores = self.fc(features)
    scores = self.bn(scores)

    return scores
```

- The context vector produced by the attention mechanism is then passed through a fully connected layer for classification.

The ClassificationHead module consists of a fully connected linear layer, followed by dropout and batch normalization. The input features are first flattened, and then passed through the linear layer to obtain the final classification scores for the four classes.

- A batch normalization layer is applied to stabilize the training.

The ImageClassifier module combines the EncoderCNN, LuongAttention, and ClassificationHead components:

```
class ImageClassifier(nn.Module):
  def __init__(self, encoder, encoder_dim, num_classes):
    super(ImageClassifier, self).__init__()
    self.encoder = encoder
    self.attention = LuongAttention(encoder_dim)
    self.classifier = ClassificationHead(encoder_dim, num_classes)

  def forward(self, images):
    features = self.encoder(images)
    # Apply attention mechanism
    context, _ = self.attention(features, features)
    scores = self.classifier(context)

    return scores
```

During the forward pass, the input images are first processed by the EncoderCNN to extract features. These features are then passed through the LuongAttention module, which computes the attended features (context vector). Finally, the attended features are passed to the ClassificationHead module, which performs the final classification and returns the scores for each class.

- Initially, the ResNet-50 weights are frozen to prevent them from being updated during the first phase of training.

- In the second phase, the entire model, including the ResNet-50 layers, is fine-tuned to improve performance.

The overall model architecture leverages the powerful feature extraction capabilities of the pretrained ResNet50, combined with the Luong Attention mechanism to focus on the most relevant features, and a Classification Head trained specifically for the corn kernel classification task.

```
==========================================================================================
==============
Layer (type:depth-idx)                             Output Shape              Param #
==========================================================================================
==============
ImageClassifier                                    [32, 4]                   --
├─EncoderCNN: 1-1                                  [32, 196, 1024]           --
│    └─Sequential: 2-1                             [32, 1024, 14, 14]        --
│    │    └─Conv2d: 3-1                            [32, 64, 112, 112]        (9,408)
│    │    └─BatchNorm2d: 3-2                       [32, 64, 112, 112]        (128)
│    │    └─ReLU: 3-3                              [32, 64, 112, 112]        --
│    │    └─MaxPool2d: 3-4                         [32, 64, 56, 56]          --
│    │    └─Sequential: 3-5                        [32, 256, 56, 56]         (215,808)
│    │    └─Sequential: 3-6                        [32, 512, 28, 28]
(1,219,584)
│    │    └─Sequential: 3-7                        [32, 1024, 14, 14]
(7,098,368)
├─LuongAttention: 1-2                              [32, 196, 1024]           --
├─ClassificationHead: 1-3                          [32, 4]                   --
│    └─Linear: 2-2                                 [32, 4]                   802,820
│    └─BatchNorm1d: 2-3                            [32, 4]                   8
==========================================================================================
==============
```

Figure 15: The CNN-Attn model processing an input of size (batch_size=32, 3, 224, 224) ,where batch_size is the number of inputs per batch, 3 the number of color channels, 224, 224 the width and height of the input images per channel, in pixels/numbers of row-columns

Vision Transformer

Transformer

Transformers are a type of neural network architecture that was initially proposed for sequence to-sequence tasks in natural language processing (NLP), such as machine translation. However, their self-attention mechanism and ability to capture long-range dependencies have made them successful in various domains, including computer vision.

The core component of a transformer is the multi-headed self-attention mechanism. Self Attention allows the model to weigh the importance of different parts of the input sequence when computing the representation of a particular part. This is achieved by calculating the dot product between the query, key, and value vectors derived from the input sequence.

In the context of NLP, the input sequence is a sequence of word embeddings or token

embeddings. In the case of computer vision, the input sequence is a sequence of image patches or feature maps.

Transformers are composed of an encoder and a decoder, both of which consist of multiple layers of multi-headed self-attention and feed-forward neural networks. The encoder processes the input sequence and generates a sequence of encoded representations, while the decoder generates the output sequence, attending to the encoded representations from the encoder.

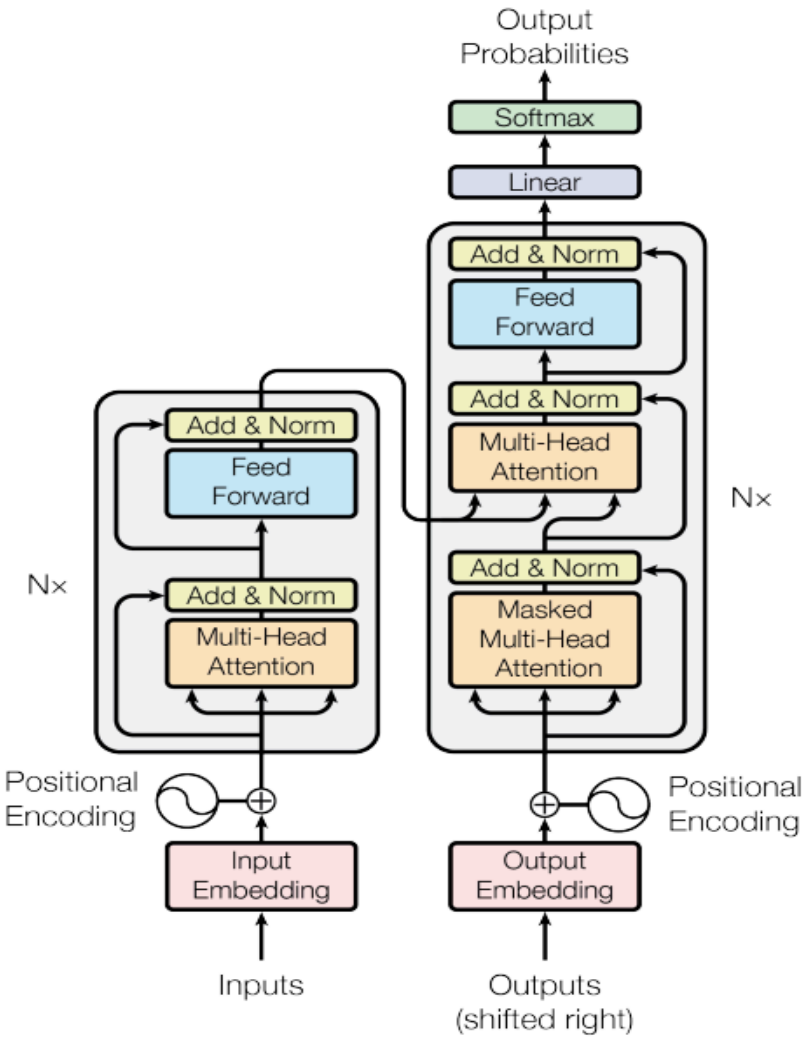

Figure 16: The Transformer model architecture

Vision Transformer (ViT)

The ViT model follows the same Transformer encoder architecture.

Unlike traditional convolutional neural networks (CNNs), which rely on convolutions and pooling operations to extract features from images, ViT treats an image as a sequence of image patches and processes it using the transformer encoder. The input image is split into a sequence of fixed-size patches, which are then linearly projected and used as input to the Transformer encoder.

The ViT architecture is as follows:

1. Patch Embedding: The input image is first divided into a sequence of fixed-size non overlapping patches. Each patch is then flattened and linearly projected into a lower dimensional embedding space using a trainable linear projection layer. This step transforms the two-dimensional image data into a one-dimensional sequence of patch embeddings, which can be processed by the transformer encoder.

2. Position Embedding: Since the transformer does not have an inherent notion of positional information, position embeddings are added to the patch embeddings to retain the positional information of each patch within the image.

3. Transformer Encoder: The sequence of patch embeddings, along with the position embeddings, is then fed into the transformer encoder. The transformer encoder consists of multiple identical layers, each comprising:

    - Multi-headed self-attention: This mechanism allows the model to weigh the importance of different patches when computing the representation of a particular patch, capturing long-range dependencies within the image.

    - Multi-layer perceptron (MLP): A feed-forward neural network applied to each patch independently, further processing the patch representations.

    - Layer normalization and residual connections: Normalization and skip connections are used to improve training stability and facilitate better gradient flow.

4. Classification Head: After the final transformer encoder layer, a special [CLS] token embedding, which was prepended to the sequence of patch embeddings, is used as a global representation of the entire image. This [CLS] token embedding is then passed through a small multi-layer perceptron (MLP) head to obtain the final classification scores.

The self-attention mechanism in ViT allows the model to capture global dependencies and contextual information within the image, while the transformer encoder's hierarchical structure enables the model to learn increasingly abstract and high-level representations of the input.

ViT has achieved competitive performance on various image classification benchmarks, demonstrating the effectiveness of transformer architectures in computer vision tasks.

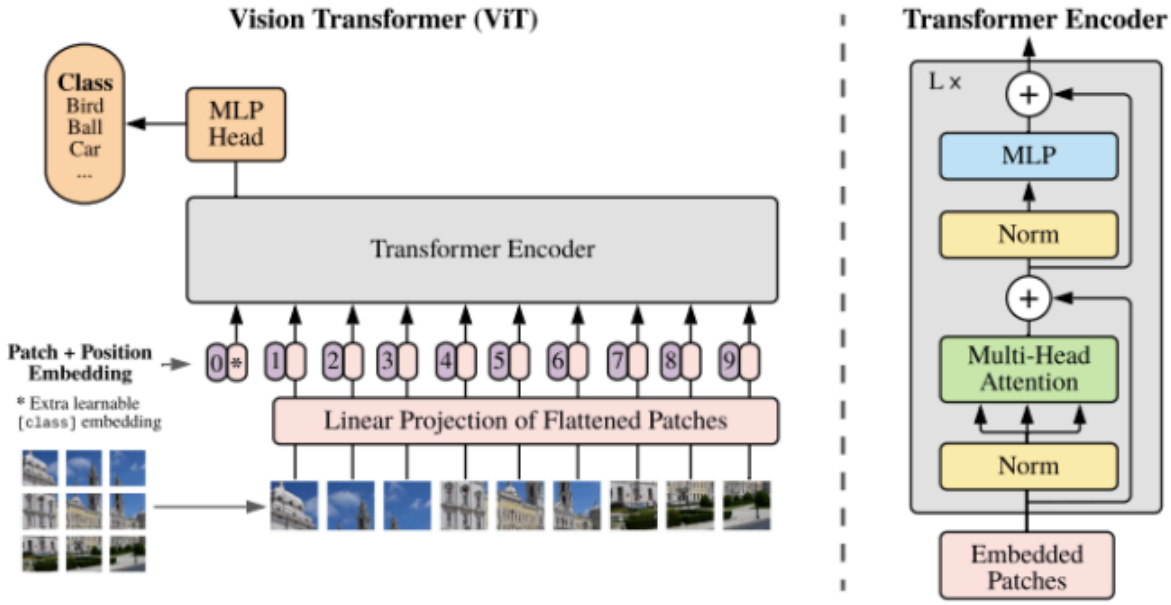

Figure 17: The Vision Transformer model overview

```
==========================================================================================
==============
Layer (type:depth-idx)                                  Output Shape              Param #
==========================================================================================
==============
ImageClassifier                                         [32, 4]                   --
├─EncoderViT: 1-1                                       [32, 197, 768]            --
│    └─VisionTransformer: 2-1                           --                        921,064
│    │    └─PatchEmbed: 3-1                             [32, 196, 768]            (590,592)
│    │    └─Dropout: 3-2                                [32, 197, 768]            --
│    │    └─Identity: 3-3                               [32, 197, 768]            --
│    │    └─Identity: 3-4                               [32, 197, 768]            --
│    │    └─Sequential: 3-5                             [32, 197, 768]
(85,054,464)
│    │    └─LayerNorm: 3-6                              [32, 197, 768]            (1,536)
├─ClassificationHead: 1-2                               [32, 4]                   --
│    └─Linear: 2-2                                      [32, 4]                   3,076
│    └─BatchNorm1d: 2-3                                 [32, 4]                   8
==========================================================================================
==============
```

Figure 18: The ViT model processing an input of size (batch_size=32, 3, 224, 224) , where batch_size is the number of inputs per batch, 3 the number of color channels, 224, 224 the width and height of the input images per channel, in pixels/numbers of row-columns.

The code for the ViT-based model is similar to the CNN-Attn-based model.

First we extract the feature representations from the input images. These feature representations are then returned for use in the classification head.

Next, we extract the [CLS] token features which serve as a global representation of the entire image. These [CLS] token features are then passed through the fully connected layer, dropout, and batch normalization to obtain the final classification scores.

```
class EncoderViT(nn.Module):
  def __init__(self, model_name='vit_base_patch16_224',
pretrained=True):
    super(EncoderViT, self).__init__()
    self.model = timm.create_model(model_name, pretrained=pretrained)
    for param in self.model.parameters():
      param.requires_grad = False

  def forward(self, images):
    with torch.no_grad():
      features = self.model.forward_features(images)
    return features

class ClassificationHead(nn.Module):
  def __init__(self, encoder_dim, num_classes):
    super(ClassificationHead, self).__init__()
    self.fc = nn.Linear(encoder_dim, num_classes)
    self.dropout = nn.Dropout(0.5)
    self.bn = nn.BatchNorm1d(num_classes)

  def forward(self, features):
    batch_size, _, _ = features.shape
    # Get the CLS token
    features = features[:, 0, :]
    scores = self.fc(features)
    scores = self.bn(scores)
    return scores

class ImageClassifier(nn.Module):
  def __init__(self, encoder, encoder_dim, num_classes):
    super(ImageClassifier, self).__init__()
    self.encoder = encoder
    self.classifier = ClassificationHead(encoder_dim, num_classes)

  def forward(self, images):
    features = self.encoder(images)
    scores = self.classifier(features)
```

```
    return scores
```

Training

- Both the ResNet50-Attn and ViT models were trained for 20 epochs with a batch size of 128.
- For the ResNet50-Attn model, the ResNet50 weights were frozen for the first 10 epochs and then unfrozen for the next 10 epochs for fine-tuning.
- For the ViT model, the weights were kept frozen since fine-tuning the large ViT model would require significant computational resources.
- The dataset was split into 80% for training (11,457 images) and 20% for validation (2,865 images).

Evaluation

Loss (Cross Entropy)

Cross-entropy loss, or log loss, measures the performance of a classification model whose output is a probability value between 0 and 1. Cross-entropy loss increases as the predicted probability diverges from the actual label. So predicting a probability of 0.012 when the actual observation label is 1 would be bad and result in a high loss value. A perfect model would have a log loss of 0.

The probability for each class, by default, is:

$$l(x, y) = L = \{l_1, \ldots, l_N\}^T, l_n = -\sum_{c=1}^{C} w_c \log \frac{exp(x_{n,c})}{\sum_{i=1}^{C} exp(x_{n,i})} y_{n,c}$$

where

- x is the input
- y is the target
- w is the weight
- C is the number of classes
- N spans the batch dimension

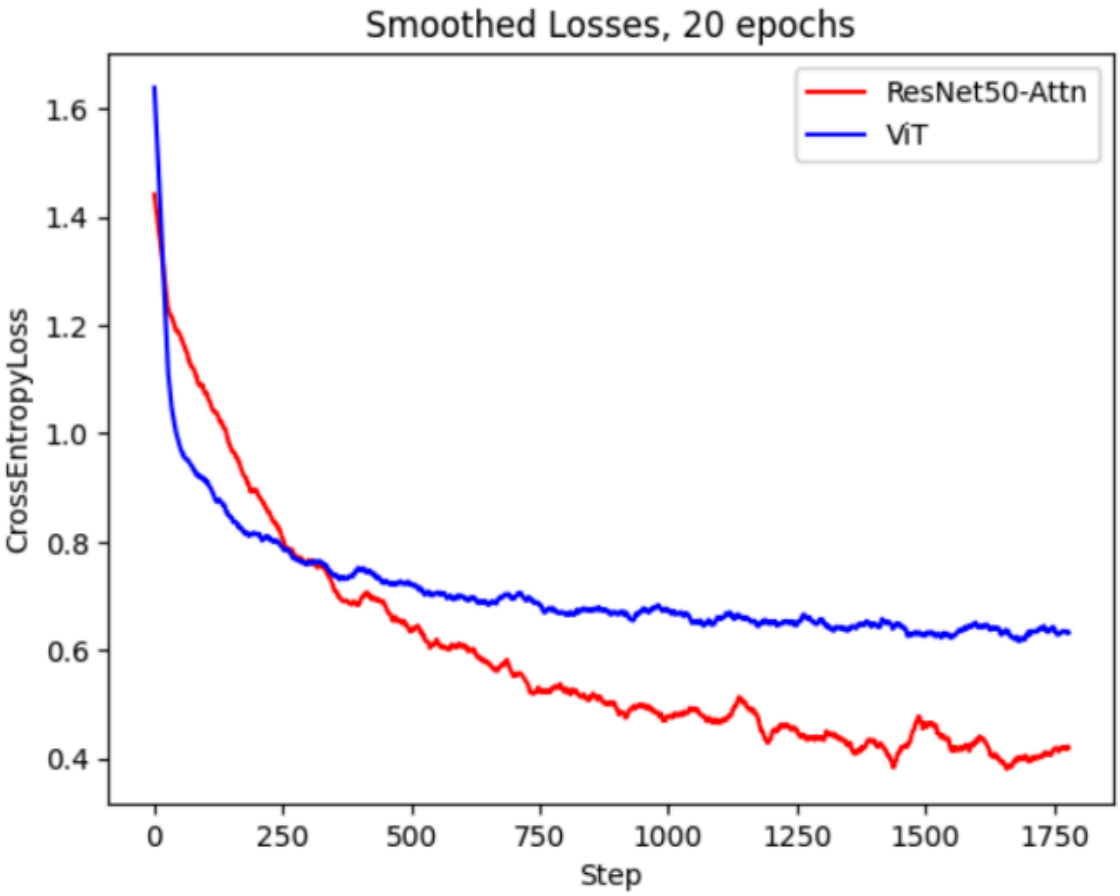

Figure 18: Smoothed CrossEntropy Loss plot, 20 epochs, resNet50-Attn and ViT

- For the ResNet50-Attn model:
    - Loss decreased to around 0.5 before fine-tuning, then further decreased to around 0.45 after fine-tuning ResNet50 weights.
    - Fine-tuning helped the model adapt better to the task, improving performance.
    - However, the slight increase in loss after epoch 20 suggests some overfitting to the training data.
    - Overfitting occurs when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data.

    - This means that the noise or random fluctuations in the training data is picked up and learned as concepts by the model. The problem is that these concepts do not apply to new data and negatively impact the model's ability to generalize.

- For the ViT model:

    - Loss decreased sharply initially to around 0.7, then stabilized without much further improvement.

    - With frozen weights, the ViT could not fine-tune to the task, limiting its ability to reduce loss further.

    - The higher final loss of 0.7 compared to 0.45 for ResNet50-Attn indicates worse overall performance.

In conclusion, both models were able to effectively learn from the training data and reduce their cross-entropy loss over time.

However, the ResNet50-Attn model achieved a lower final loss compared to the ViT model, possibly due to the fine-tuning of the ResNet50 weights.

On the other hand, the ViT model achieved a respectable performance despite its weights being kept frozen, indicating that the pre-trained ViT weights were already quite effective for this task.

However, both models might benefit from further optimization to reduce overfitting and improve their ability to generalize to new data. This could involve techniques such as regularization, early stopping, or dropout.

Additionally, collecting more diverse training data, using data augmentation techniques, or even using synthetic data could also help to improve the models' performance.

Accuracies

1. ResNet50-Attn 10 (before unfreezing): 0.7124
2. ResNet50-Attn 20 (after unfreezing): 0.7068
3. ViT 10: 0.7190
4. ViT 20: 0.7211

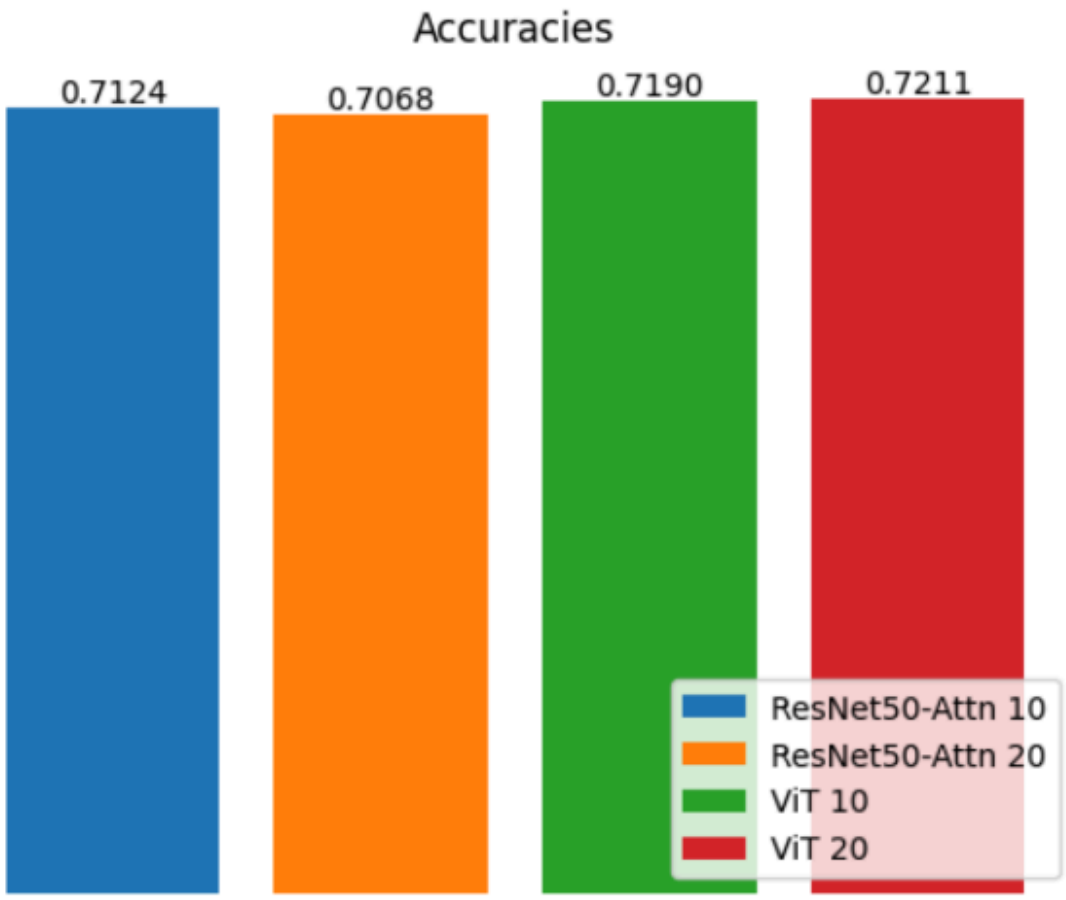

Figure 19: Accuracies of each model

ResNet50-Attn accuracy slightly decreased from 71.24% to 70.68% after fine-tuning, likely due to some overfitting.

ViT started slightly higher at 71.9% and increased to 72.11%, continuing to learn through more training epochs.

In conclusion, both models achieved similar performance, with the ViT model slightly outperforming the ResNet50-Attn model in terms of overall accuracy on the validation set. However, both models had difficulty distinguishing between certain classes, which could be an area for further investigation and improvement.

Confusion Matrix

A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known. It allows visualization of the performance of an algorithm.

- True Positives (TP): These are cases in which we predicted yes (the crop is 'silkcut'), and the crop is actually 'silkcut'.
- True Negatives (TN): We predicted no, and the crop is not 'silkcut'.
- False Positives (FP): We predicted yes, but the crop isn't 'silkcut'. (Also known as a "Type I error.")
- False Negatives (FN): We predicted no, but the crop is 'silkcut'. (Also known as a "Type II error.")

The confusion matrices provide insight into the model's performance by showing the distribution of correct and incorrect predictions for each class. Higher values along the diagonal indicate correct predictions, while off-diagonal values represent misclassifications.

1. ResNet50-Attn 10: 0.7124

```
[[645 135 72 31]
[192 934 54 15]
[127 49 281 24]
[ 51 48 26 181]]

[[0.73046433 0.15288788 0.0815402 0.03510759]
[0.16066946 0.78158996 0.04518828 0.0125523 ]
[0.26403326 0.1018711 0.58419958 0.04989605]
```

[0.16666667 0.15686275 0.08496732 0.59150327]]

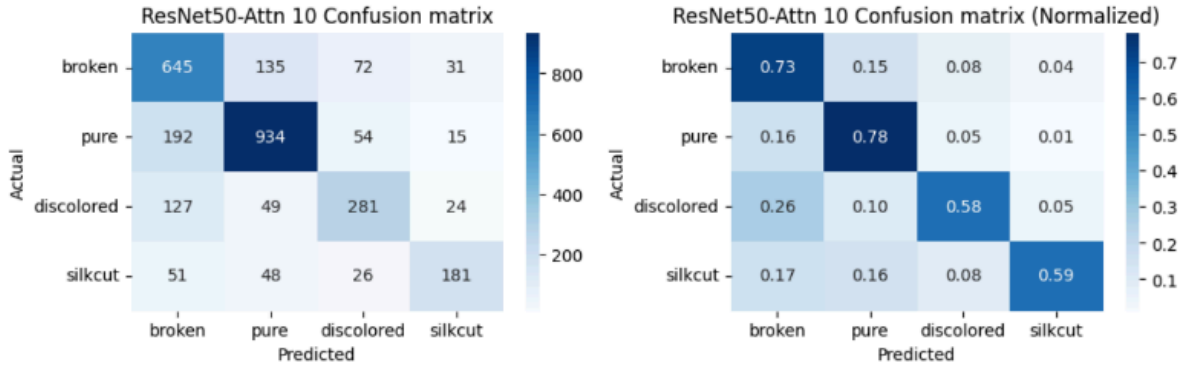

Figure 20: Confusion matrix of ResNet50-Attn 10 epochs - Count(left), Normalized(Right)

Overall accuracy of 71.24%

Actuals of each class:

- Broken: 645 (73%)
- Pure: 934 (78%)
- Discolored: 281 (58%)
- Silkcut: 181 (59%)

Notably:

- Relatively balanced performance across Broken (73% accuracy) and Pure (78%) classes
- However, struggles significantly with minority Discolored (58%) and Silkcut (59%)
- 26% of Discolored samples misclassified as Broken
- 17% of Silkcut misclassified as Broken, 16% as Pure

This Indicates the model has difficulty distinguishing the more subtle defects from the visually similar Broken class.

2. ResNet50-Attn 20: 0.7068

[[601 182 63 37]
[133 1022 18 22]
[138 87 208 48]
[43 56 13 194]]

[[0.6806342 0.20611552 0.07134768 0.0419026]
[0.11129707 0.85523013 0.01506276 0.01841004]
[0.28690229 0.18087318 0.43243243 0.0997921]
[0.14052288 0.18300654 0.04248366 0.63398693]]

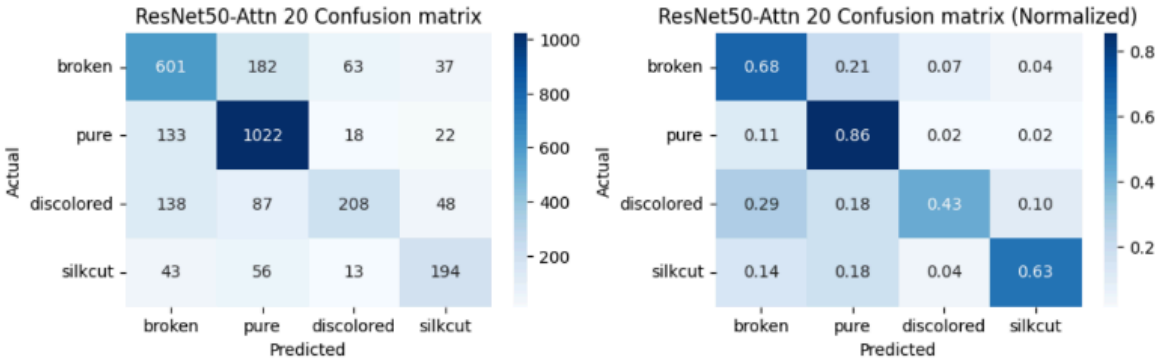

Figure 21: Confusion matrix of ResNet50-Attn 20 epochs - Count (left), Normalized (Right)

Overall accuracy drops slightly to 70.68%

Actuals of each class:

- Broken: 601 (73%)
- Pure: 1022 (86%)
- Discolored: 208 (43%)
- Silkcut: 194 (63%)

Notably:

- Broken accuracy decreases to 68%
- Pure accuracy increases to 86%, but at expense of other classes
- Discolored accuracy drops sharply to 43%

- 21%, 18%, 18% of Broken, Discolored and Silkcut respectively, now misclassified as Pure

This indicates the fine-tuning caused overfitting towards the majority Pure class

The increased training caused the model to latch onto the Pure characteristics while losing separability for defects

3. ViT 10: 0.7190

[[630 158 65 30]
[144 994 37 20]
[124 81 244 32]
[51 38 25 192]]

[[0.71347678 0.17893545 0.07361268 0.03397508]
[0.12050209 0.83179916 0.03096234 0.0167364]
[0.25779626 0.16839917 0.50727651 0.06652807]
[0.16666667 0.12418301 0.08169935 0.62745098]]

Overall accuracy of 71.9%, slightly higher than initial ResNet50-Attn

Actuals of each class:

- Broken: 630 (71%)
- Pure: 994 (83%)
- Discolored: 244 (51%)
- Silkcut: 192 (63%)

Notably:

- Pure and Silkcut saw an increase in accuracy (78% to 86% and 59% to 63%)
- But Broken, Discolored, Silkcut all decrease in accuracy
- Similar struggles with Discolored (51%) and Silkcut (63%)
- Mis-classifies Broken and Discolored as Pure more than ResNet50-Attn 10 (from 15% to 10% for ResNet, 18% to 17% for ViT)

This indicates the frozen ViT weights may have some bias towards normal "Pure" kernels

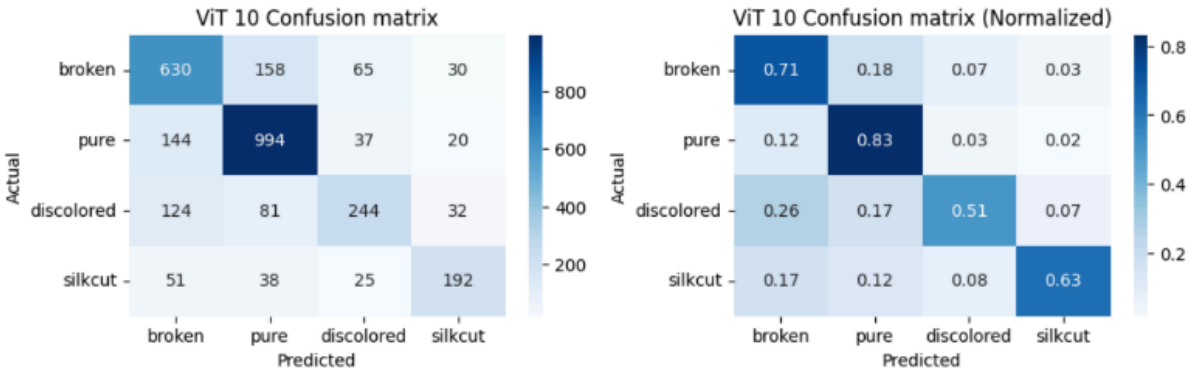

Figure 22: Confusion matrix of ViT 10 epochs - Count (left), Normalized (Right)

4. ViT 20: 0.7211

[[574 220 53 36]
[90 1060 36 9]
[99 109 244 29]
[33 61 24 188]]

[[0.65005663 0.24915062 0.06002265 0.0407701]
[0.07531381 0.88702929 0.03012552 0.00753138]
[0.20582121 0.22661123 0.50727651 0.06029106]
[0.10784314 0.19934641 0.07843137 0.61437908]]

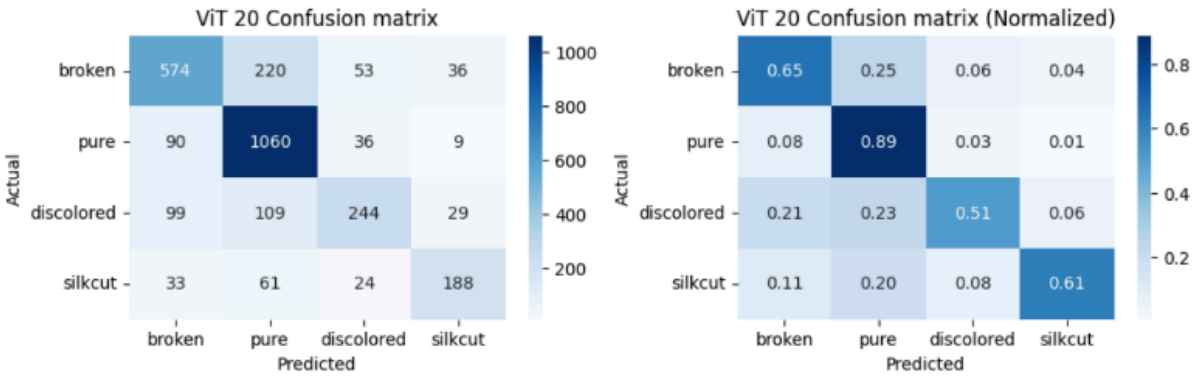

Figure 23: Confusion matrix of ViT 20 epochs - Count (left), Normalized (Right)

Overall accuracy increases marginally to 72.11%

Actuals of each class:

- Broken: 574 (65%)
- Pure: 1060 (89%)
- Discolored: 244 (51%)
- Silkcut: 188 (61%)

Notably:

- Pure accuracy continues increasing to 89%
- But Broken, Discolored, Silkcut all decrease in accuracy
- Discolored still stuck at 51% accuracy

The ViT is becoming increasingly biased towards classifying any non-pristine kernel as Pure or Broken, more than ResNet50-Attn

This suggests some overfitting despite lack of fine-tuning

In conclusion:

- Both models make strides in accuracy on the majority classes through training

- Struggle with the minority Discolored and Silkcut defect classes.

- Fine-tuning provides a boost for ResNet50-Attn initially, but leads to overfitting issues later on.

- ViT performed similarly to the early ResNet50-Attn, showing a propensity to default to classifying ambiguous cases as Pure as it trains longer on the imbalanced data.

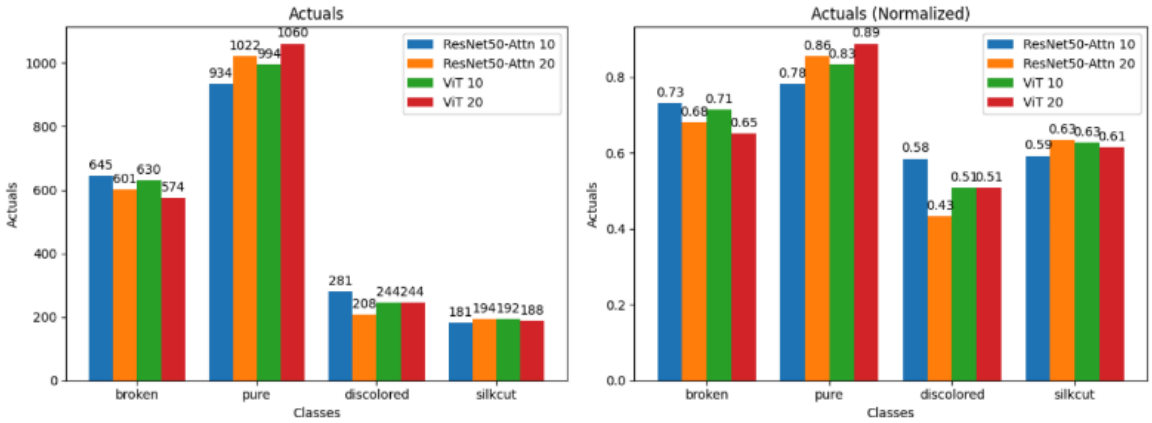

Figure 24: Actuals of all models - Count (left), Normalized (Right)

Exploration

ResNet50 Feature maps

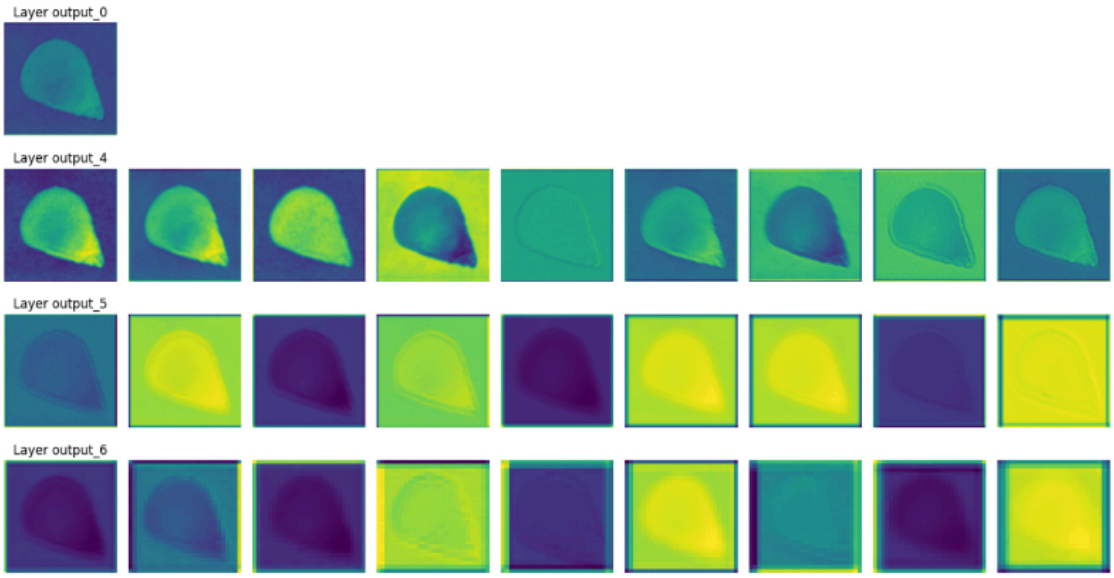

Figure 25: Mean-Feature maps of each Conv Layer in ResNet50 10 (Before finetuning) (A Pure sample)

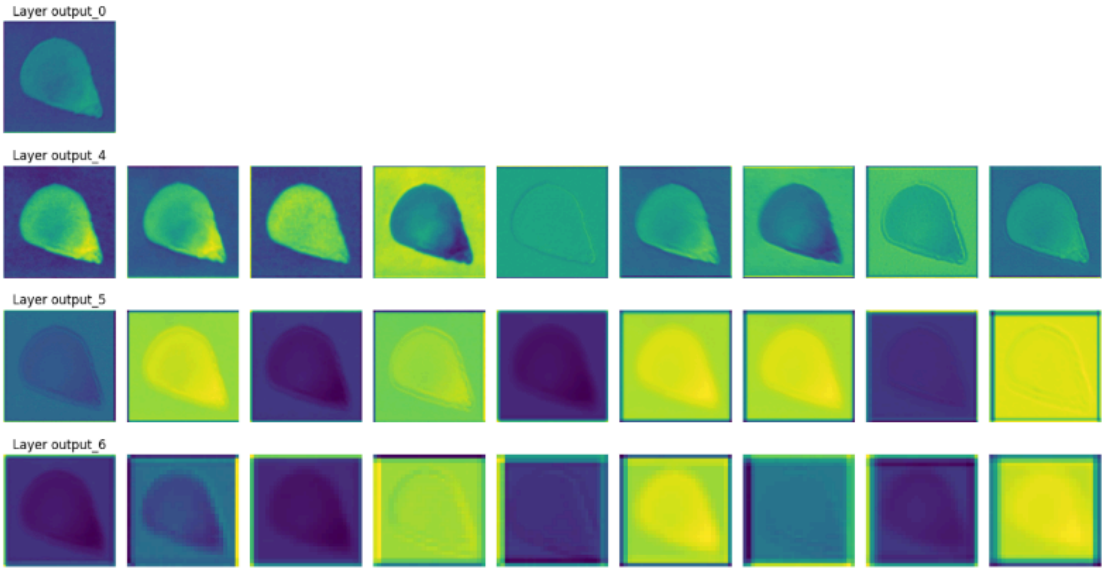

Figure 26: Mean-Feature maps of each Conv Layer in ResNet50 20 (After finetuning) (A Pure sample)

Slight changes in layer output_6 between the two models

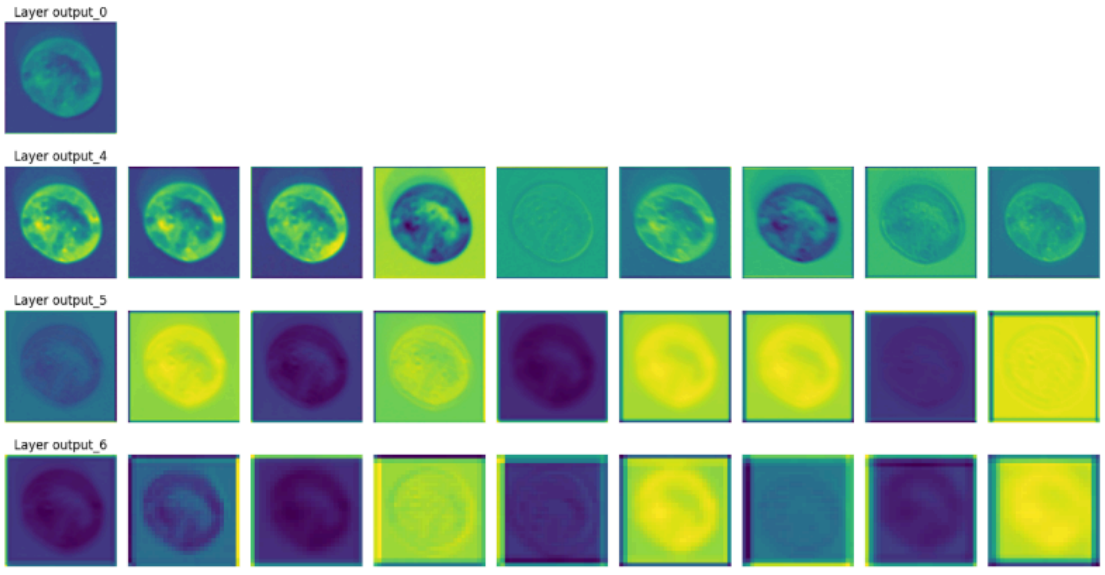

Figure 27: Mean-Feature maps of each Conv Layer in ResNet50 20 (After finetuning) (A Broken sample)

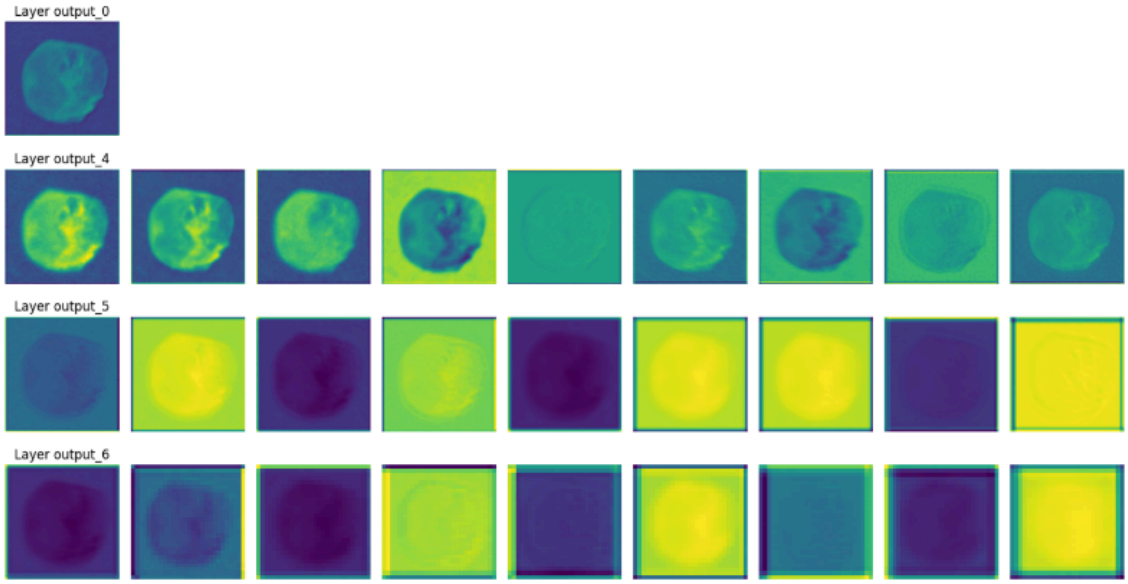

Figure 28: Mean-Feature maps of each Conv Layer in ResNet50 20 (After finetuning) (A Discolored sample)

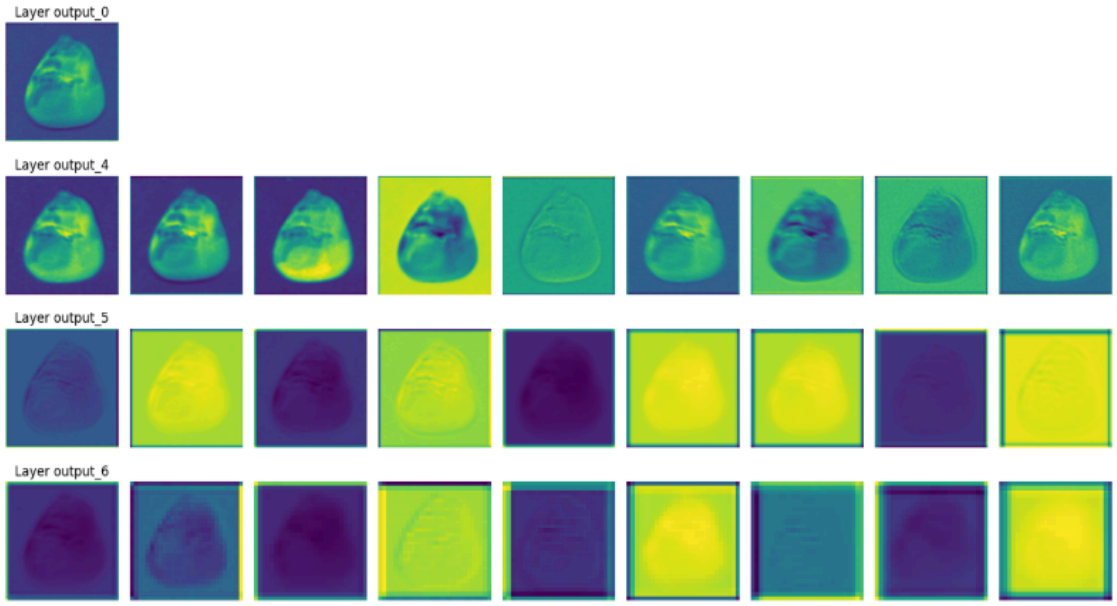

Figure 29: Mean-Feature maps of each Conv Layer in ResNet50 20 (After finetuning) (A Silkcut sample)

ResNet50-Attn Attention maps

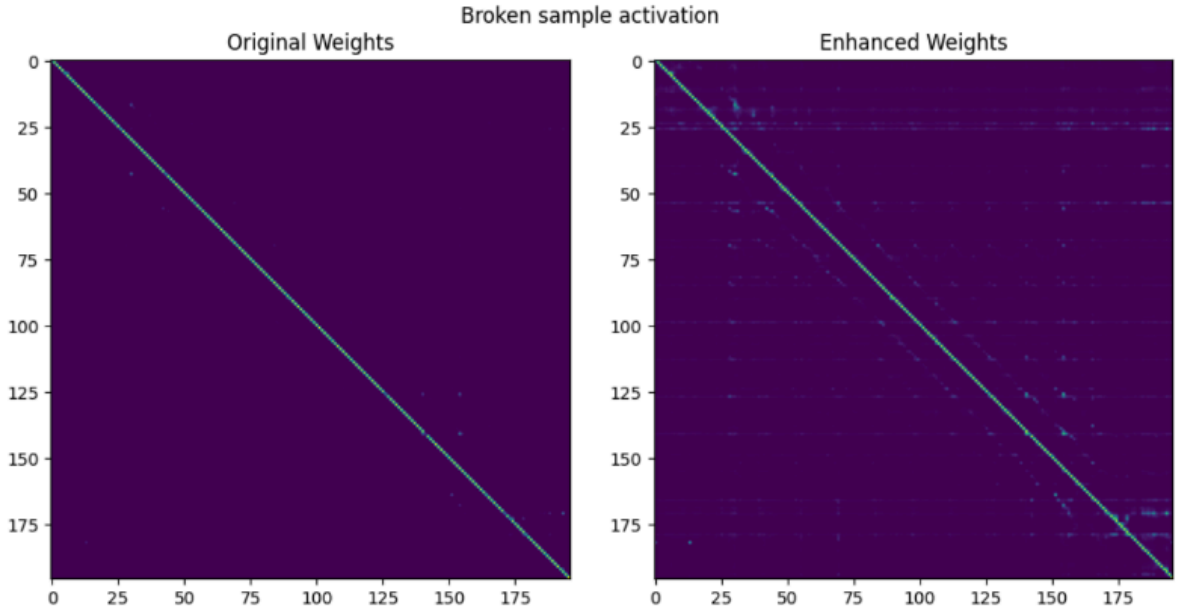

Figure 30: Luong Attention (Dot) weights (Original and Enhanced) of ResNet50 20 (After finetuning) (A Broken sample)

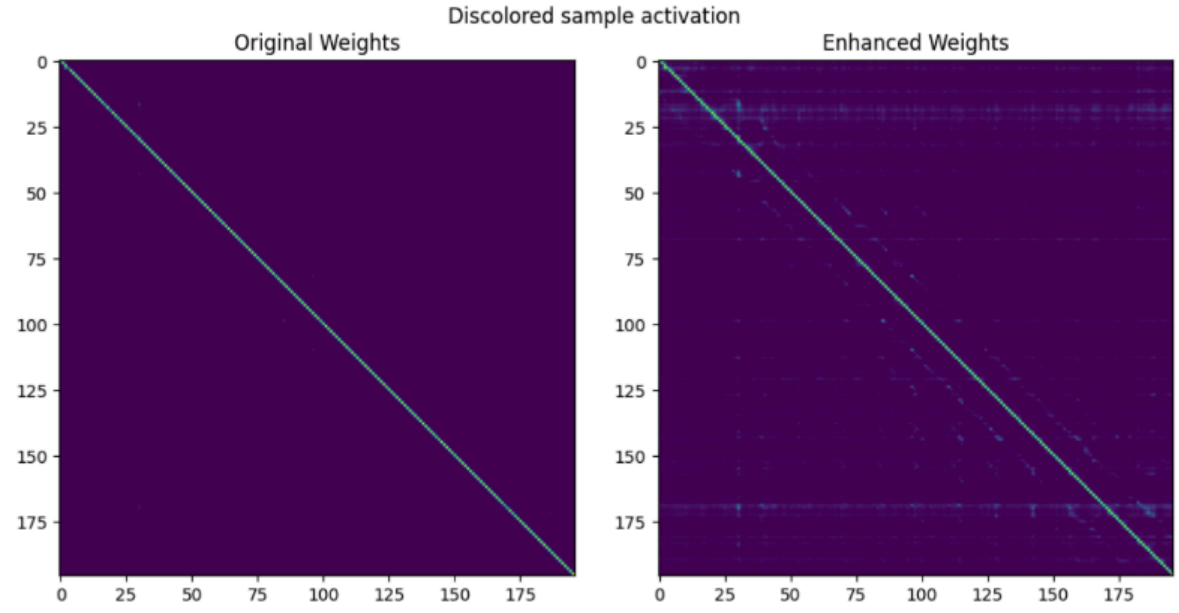

Figure 31: Luong Attention (Dot) weights (Original and Enhanced) of ResNet50 20 (After finetuning) (A Discolored sample)

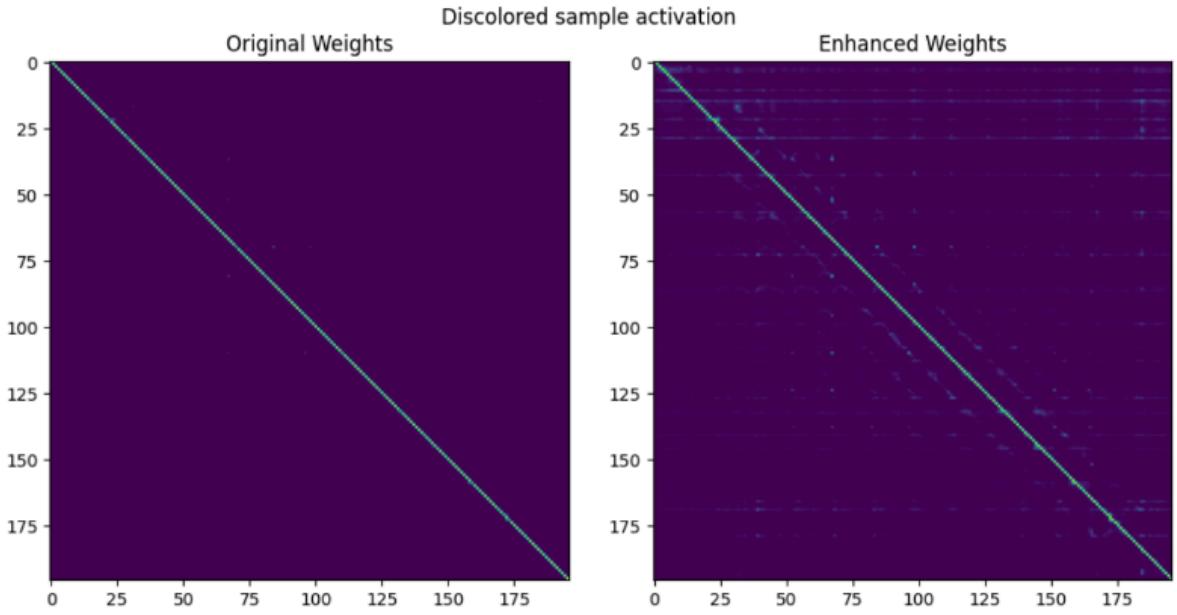

Figure 32: Luong Attention (Dot) weights (Original and Enhanced) of ResNet50 20 (After finetuning) (A Pure sample)

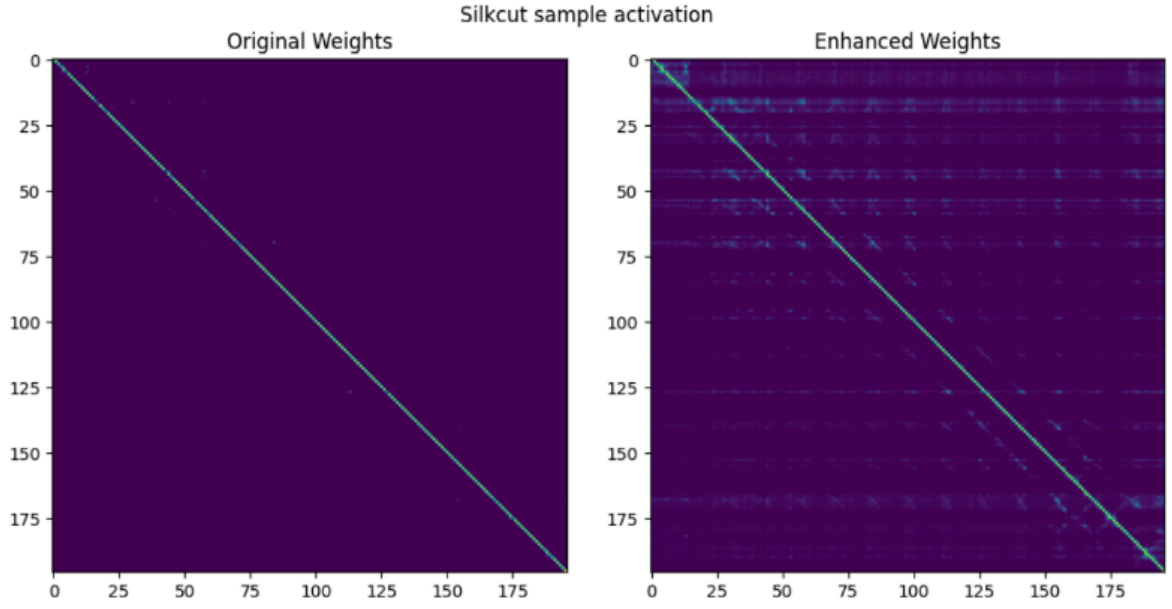

Figure 33: Luong Attention (Dot) weights (Original and Enhanced) of ResNet50 20 (After finetuning) (A Silkcut sample)

ResNet50-Attn Final conv block's Activation maps (GRADCAM)

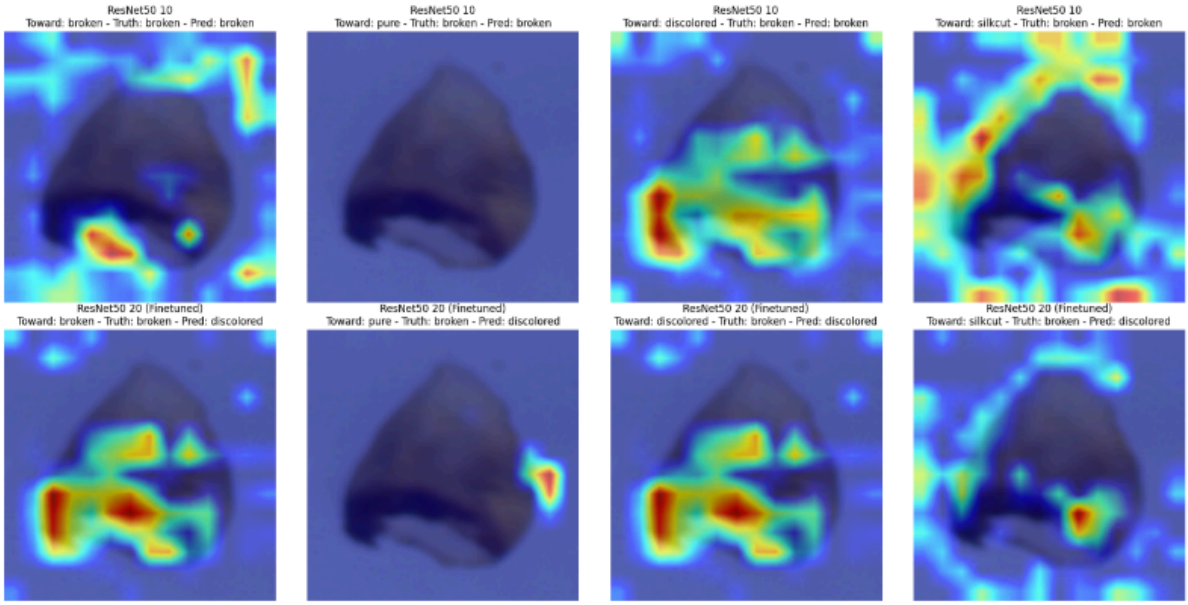

Figure 34: ResNet50 10 (top) and 20 (bottom) Activation map toward each class (Broken sample, correct and incorrect prediction)

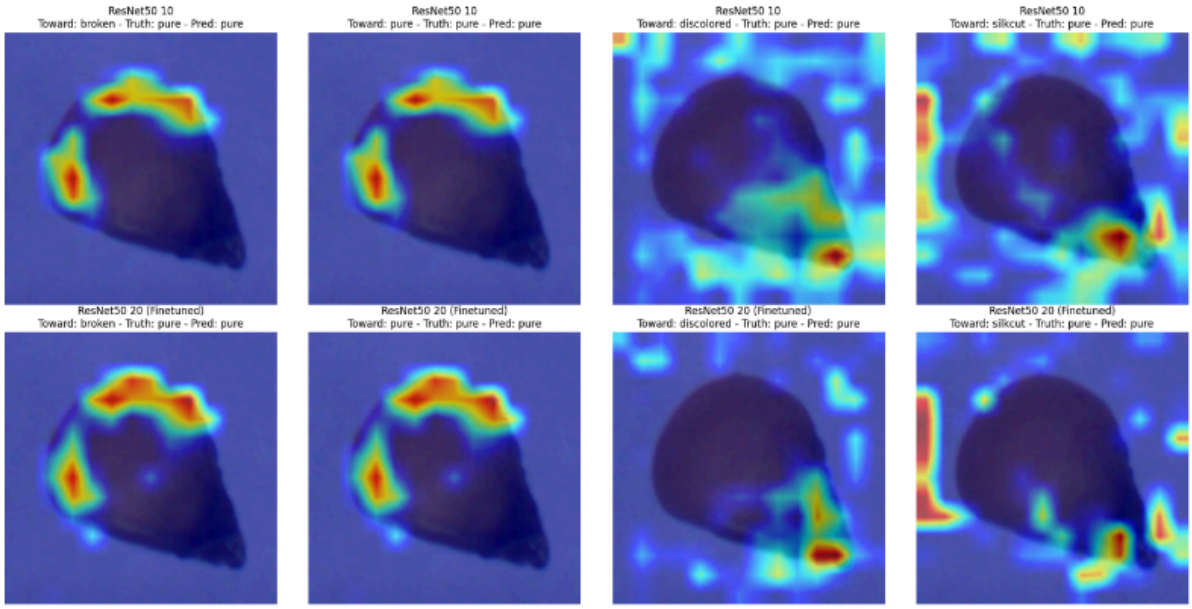

Figure 35: ResNet50 10 (top) and 20 (bottom) Activation map toward each class (Pure sample, both are correct)

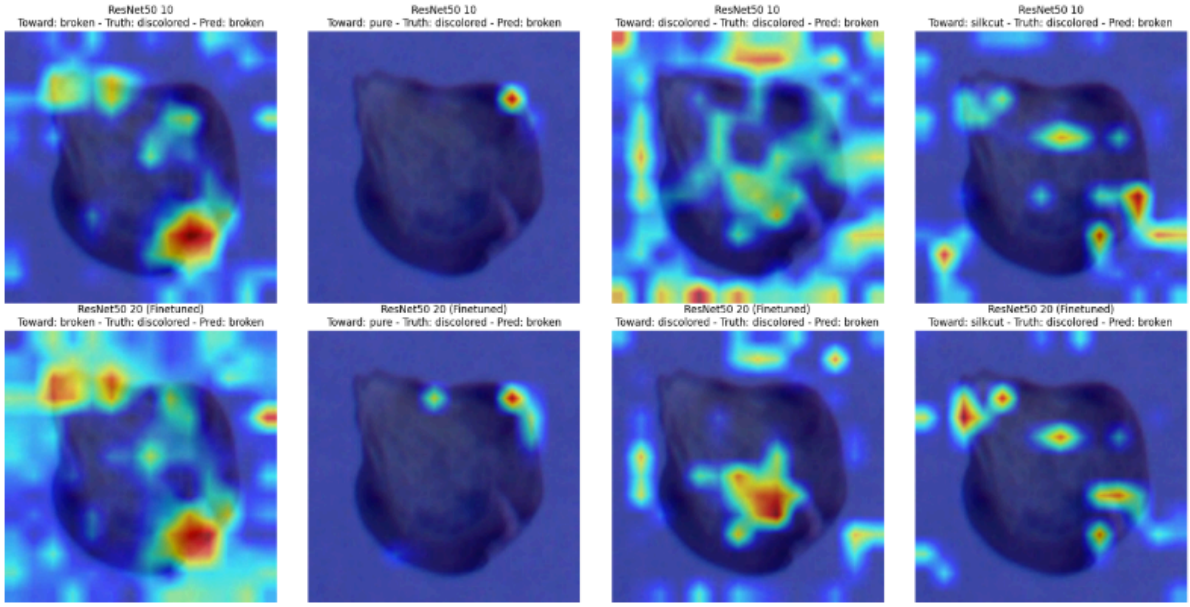

Figure 36: ResNet50 10 (top) and 20 (bottom) Activation map toward each class (Discolored sample, both are wrong, as Broken)

Note:

- 10 epochs model's output: [0.7915, 0.1412, -0.4019, -1.0206]
- 20 epochs model's output: [0.5819, -0.0548, 0.0341, -1.5741]

In order: ['broken', 'pure', 'discolored', 'silkcut']

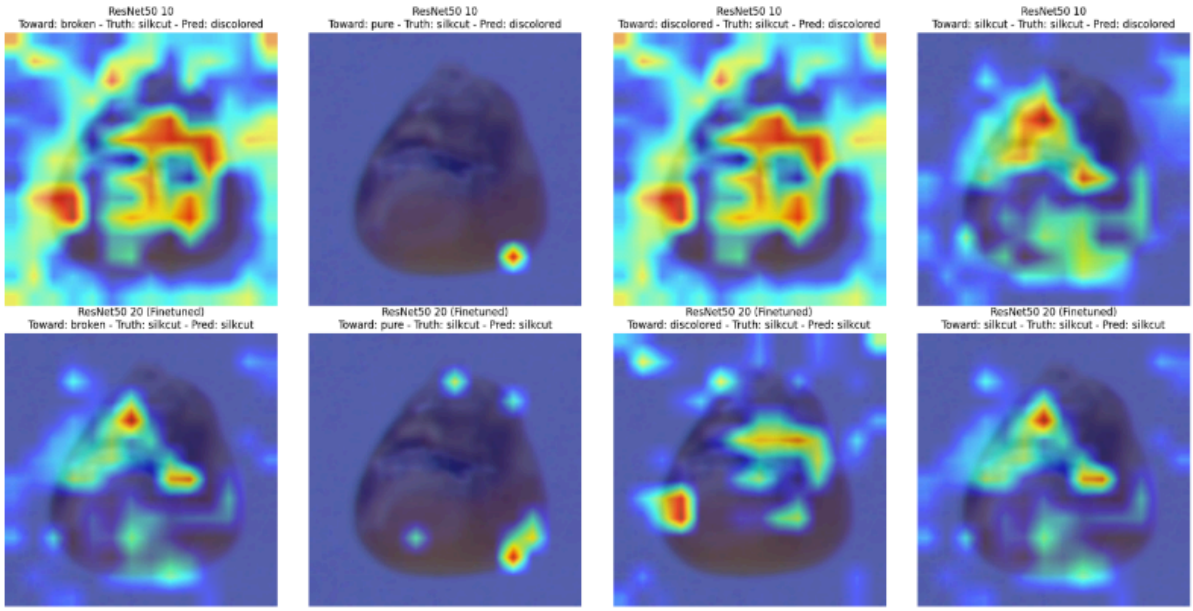

Figure 37: ResNet50 10 (top) and 20 (bottom) Activation map toward each class (Silkcut sample, incorrect and correct)

Note: 20 epochs model's output: [-1.0844, -2.2891, 0.4935, 1.4055]

ViT Activation maps (GRADCAM)

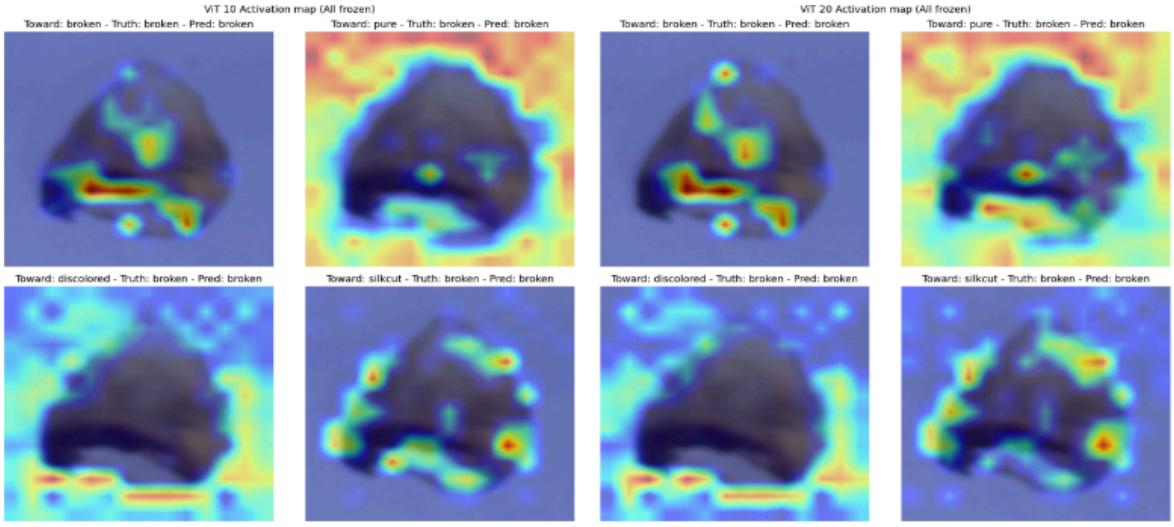

Figure 38: ViT 10 (left) and 20 (right) Activation map toward each class (Broken sample, correct and incorrect prediction)

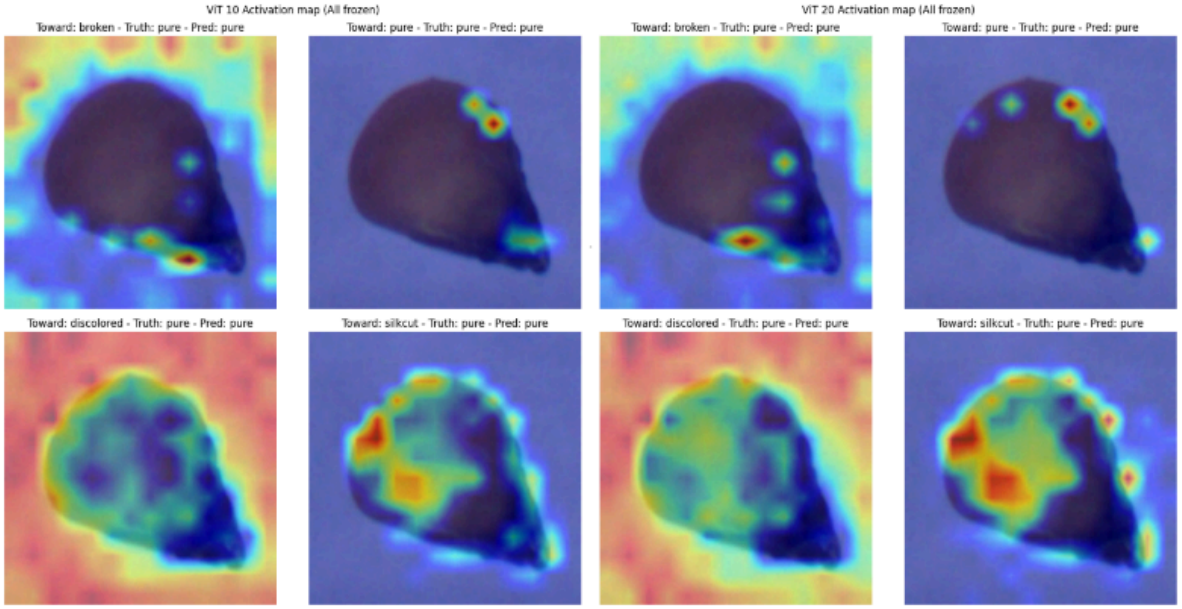

Figure 39: ViT 10 (left) and 20 (right) Activation map toward each class (Pure sample, both are correct)

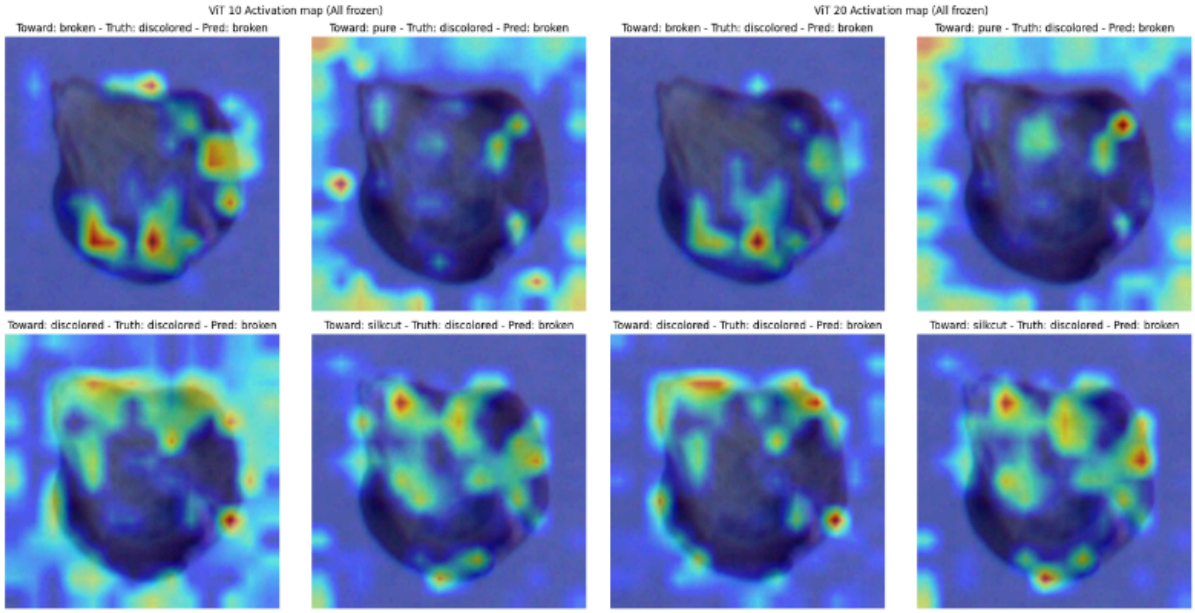

Figure 40: ViT 10 (left) and 20 (right) Activation map toward each class (Discolored sample, both are wrong, as Broken)

Note:

- 10 epochs model's output: [3.2533, -1.1244, 0.3970, -0.6474]
- 20 epochs model's output: [2.9734, -0.0654, 1.3075, -0.7734]

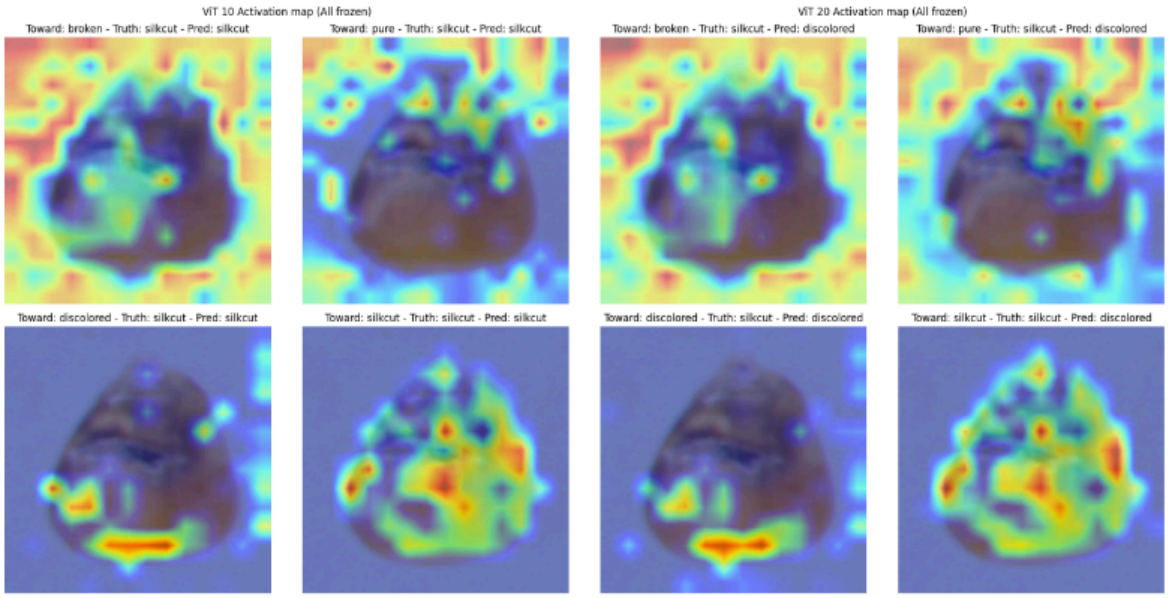

Figure 41: ViT 10 (left) and 20 (right) Activation map toward each class (Silkcut sample, incorrect and correct)

Conclusion

1. The ViT model slightly outperformed ResNet50-Attn overall (72.11% vs 70.68%), likely due to its self-attention capabilities.

2. However, both models had significant difficulty on the minority Discolored and Silkcut classes.

3. Fine-tuning provided a boost for ResNet50-Attn, but this was partially offset by overfitting issues.

4. The ViT performed respectably even with frozen weights, showing the benefit of large-scale pretraining.

To improve further, techniques are needed to address the class imbalance issue and prevent overfitting, such as:

1. Class re-balancing via oversampling minority classes or undersampling majority classes

2. Data augmentation to increase diversity

3. Stronger regularization methods like dropout, early stopping etc.

4. Additional data synthesis for more training data

Overall, this project demonstrates the applicability of deep learning models, particularly vision transformers and convolutional neural networks, to fine-grained image classification tasks like corn kernel defect detection. However, addressing class imbalance, improving minority class performance, and optimizing model architectures remain areas for future research and development.