Department of Electronic and Telecommunication Engineering

University of Moratuwa

EN3150: Pattern Recognition

Assignment 3:

Simple convolutional neural network to perform classification.

Group DeepTrace

Anjula M.K	210368V
Meemanage N.A	210385U
Shamal G.B.A	210599E
Thennakoon T.M.K.R	210642G

Contents

1.CNN for image classification

In this project, we are working with an image classification dataset from the UCI Machine Learning Repository, focused on the classification of various insect pests commonly found in jute crops. This dataset consists of 17 distinct classes, each representing a different pest, such as Beet Armyworm, Black Hairy, Cutworm, and Jute Aphid, among others. The classes are labeled from 0 to 16, each corresponding to a unique pest type.

The dataset includes 7,235 instances and features 17 categorical attributes, making it suitable for classification tasks in the area of biology. The data has been pre-divided into three partitions: training, validation, and testing, allowing for systematic model training, tuning, and evaluation. This dataset will be ideal for constructing and testing a convolutional neural network (CNN) model, aiming to accurately classify the insect types based on their images.

1.3

```
Found 6443 files belonging to 17 classes.
Found 413 files belonging to 17 classes.
Found 379 files belonging to 17 classes.
```

The dataset for this project has been effectively organized into three main partitions: training, validation, and test sets. This partitioning is essential for developing, tuning, and evaluating a robust image classification model. The breakdown is as follows:

- Training Set: Contains 6,443 images, representing all 17 insect classes. This set will be used to train the convolutional neural network (CNN), allowing the model to learn distinguishing features of each class.
- Validation Set: Contains 413 images, also covering all 17 classes. This set allows for model validation during training, enabling tuning of hyperparameters and monitoring of performance to avoid overfitting.
- Test Set: Contains 379 images, with images from each of the 17 classes. This independent set is reserved for the final evaluation of the model's accuracy and generalization ability once training is complete.

Having each class represented in all three sets ensures that the model learns patterns across all categories and can generalize well.

1.4

We designed a convolutional neural network (CNN) model to classify images into 17 distinct classes, using TensorFlow and Keras.

a.Input Layer:

- A Convolutional layer with 32 filters (3x3 kernel) and ReLU activation processes input images of size (224, 224, 3) to extract low-level features, followed by Batch Normalization for consistent input distribution.

b.Max Pooling Layer:

- A MaxPooling layer with a 2x2 window reduces the spatial dimensions of the feature maps, decreasing computational complexity and capturing spatial hierarchies.

c.Second Convolutional Layer:

- Another Convolutional layer with 64 filters (3x3 kernel) and ReLU activation extracts more complex features, followed by Batch Normalization.

d.Second Max Pooling Layer:

- A second MaxPooling layer (2x2 window) further reduces feature map dimensions, allowing the model to capture higher-level features.

e.Dropout Layer:

- A Dropout layer with a 0.5 rate mitigates overfitting by randomly dropping 50% of neurons during training.

f.Flattening Layer:

- A Flatten layer converts 2D feature maps into a 1D vector for the fully connected layers.

g.Fully Connected (Dense) Layer:

- A Dense layer with 128 units and ReLU activation learns high-level feature combinations, with L2 regularization (0.001) to reduce overfitting. Batch Normalization is also applied.

h.Second Dropout Layer:

- Another Dropout layer (0.5 rate) enhances generalization and reduces overfitting.

i.Output Layer:

- The final Dense layer with 17 units and SoftMax activation converts outputs into probabilities for 17 classes.

1.5

a.Convolutional Layers:

1. First Convolutional Layer:
    - Activation Function: ReLU
    - Kernel Size: (3, 3)
    - Number of Filters: 32
    - Input Shape: (224, 224, 3)
2. Second Convolutional Layer:
    - Activation Function: ReLU
    - Kernel Size: (3, 3)
    - Number of Filters: 64

b.Max Pooling Layers:

1. First Max Pooling Layer:
    - Pool Size: (2, 2)
2. Second Max Pooling Layer:
    - Pool Size: (2, 2)

c.Dropout Layers:

1. First Dropout Layer:
    - Dropout Rate: 0.5
2. Second Dropout Layer:
    - Dropout Rate: 0.5

d.Fully Connected (Dense) Layer:

1. Dense Layer:
    - Number of Units: 128
    - Activation Function: ReLU
    - Regularization: L2 with a factor of 0.001

e.Output Layer:

1. Output Dense Layer:

- Number of Units: 17
- Activation Function: Softmax

1.6

a. ReLU (Rectified Linear Unit) Activation Function:

In the convolutional layers, we use the ReLU (Rectified Linear Unit) activation function because it introduces non-linearity while maintaining computational efficiency. This allows the model to learn complex patterns effectively. The ReLU function outputs zero for negative values and returns the input value for positive values, which helps mitigate the vanishing gradient problem during training. Its simplicity and efficiency make it a widely adopted choice in convolutional neural networks (CNNs).

Also, the non-saturating nature of ReLU helps in faster convergence compared to traditional activation functions like sigmoid or tanh, especially in deeper networks.

b. Softmax Activation Function:

In the output layer, we use the Softmax activation function because it is ideal for multi-class classification problems, particularly as this model has 17 classes. Softmax transforms the raw output logits into probabilities for each class, ensuring that the sum of these probabilities equals one. This property is essential for interpreting the model's outputs as class probabilities, enabling effective decision-making based on the highest probability class.

Also, Softmax provides a clear interpretation of the model's confidence in each class, allowing for better decision-making based on the highest probability output.

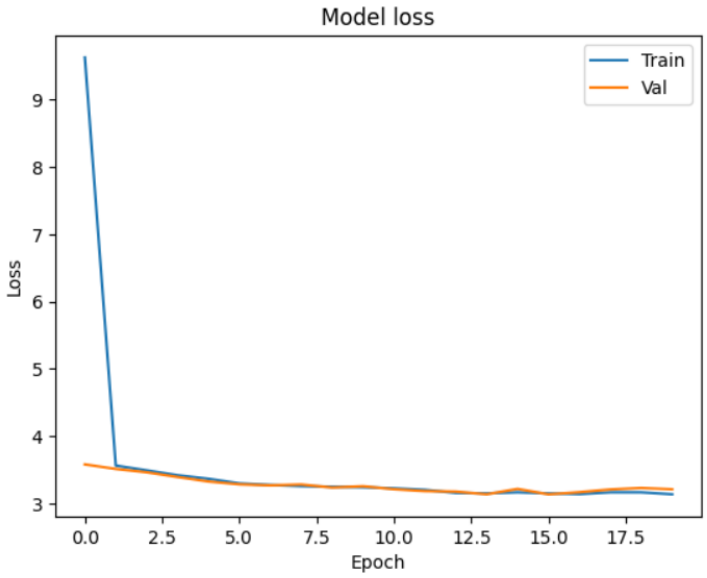
1.7

A convolutional neural network (CNN) was designed with 3 convolutional layers, max pooling, dropout, and fully connected layers to perform feature extraction and classification. The model was compiled using the Adam optimizer, sparse categorical cross-entropy as the loss function, and accuracy as the evaluation metric. It was trained for 20 epochs using the training dataset, with validation conducted on the validation dataset. Throughout the training process, both training and validation loss and accuracy were monitored to evaluate the model's performance.

Results Observed:

- Training Accuracy: Gradually improved over epochs, reaching around 12%.
- Validation Accuracy: Remained low (~18-19%), showing minimal improvement.
- Validation Loss: Decreased slightly but fluctuated around 3.2, indicating that the model struggled to generalize well to the validation dataset.

The training and validation loss over the epochs were plotted using Matplotlib to visualize the model's performance and assess its learning behavior during training and validation.



1.8

In our model with convolutional and dense layers, Adam's adaptive learning rates efficiently adjusted parameter updates for different layers, whereas SGD would require careful manual tuning, potentially delaying the training process.

Adam's built-in momentum accelerated convergence during your 20-epoch training period, helping to stabilize parameter updates, whereas SGD without momentum might have struggled to make significant progress in the same timeframe.

The training process showed fluctuating validation loss, which Adam managed well with its robust gradient adjustments, while SGD might have been more sensitive to these fluctuations, leading to unstable training.

Also, Adam converged more quickly, reducing the training loss consistently within 20 epochs, while SGD would likely have needed more epochs to achieve similar loss values due to its slower convergence.

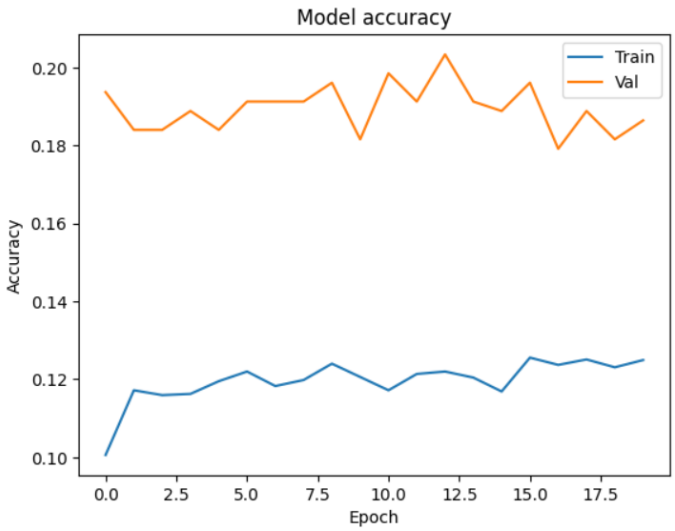
1.9

In our dataset, the class labels are provided as integers (0, 1, 2, ... 16 for 17 classes). Sparse categorical cross-entropy directly handles these integer labels without requiring them to be one-hot encoded, simplifying the input preparation.

Model is designed for a multi-class classification task (17 classes), where each input belongs to exactly one class. Sparse categorical cross-entropy is specifically designed for such scenarios.

Sparse categorical cross-entropy is computationally efficient because it avoids the additional step of converting integer labels to one-hot encoded vectors, reducing memory and processing overhead.

1.10




```
Confusion Matrix:
[[ 0  0  0  0  1  0  0  1  0 15  0  2  0  0  0  1  0]
 [ 0  0  0  0  0  0  0  0  0 20  0  0  0  0  0  0  0]
 [ 0  0  0  0  1  0  0  0  0 13  0  4  0  0  0  2  0]
 [ 0  0  0  0  1  0  0  0  0 18  0  1  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  1  0 19  1  1  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  1  0 19  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0 23  0  1  0  0  0  0]
 [ 0  0  0  0  1  0  0  1  0 19  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  1  0 22  0  1  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  1  0 21  0  1  0  0  0  1  0]
 [ 0  0  0  0  2  1  0  0  0  0 21  0  0  0  0  0  0  0]
 [ 0  0  0  0  1  0  0  0  1  0 17  1  3  0  0  0  1  0]
 [ 0  0  0  0  0  0  0  0  0  0 23  0  1  0  0  0  0  0]
 [ 0  0  0  0  2  0  0  0  0  0 21  0  1  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  1  0 17  0  0  0  0  0  2  0]
 [ 0  0  0  0  0  0  0  0  0  0 24  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0 22  0  1  0  0  0  1  0]]

Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        20
           1       0.00      0.00      0.00        20
...
    accuracy                           0.07       379
   macro avg       0.02      0.06      0.02       379
weighted avg       0.02      0.07      0.02       379
```

Training and Validation Accuracy:

- The training accuracy remained low, around 12%, across the 20 epochs.
- The validation accuracy fluctuated slightly but stayed around 18–20%, indicating that the model struggled to generalize to the validation dataset.

Confusion Matrix:

- The confusion matrix shows that the model failed to correctly classify most classes, with significant misclassifications and very few correct predictions.
- This indicates poor performance across all 17 classes, suggesting the model could not learn meaningful patterns from the data.

Classification Summary:

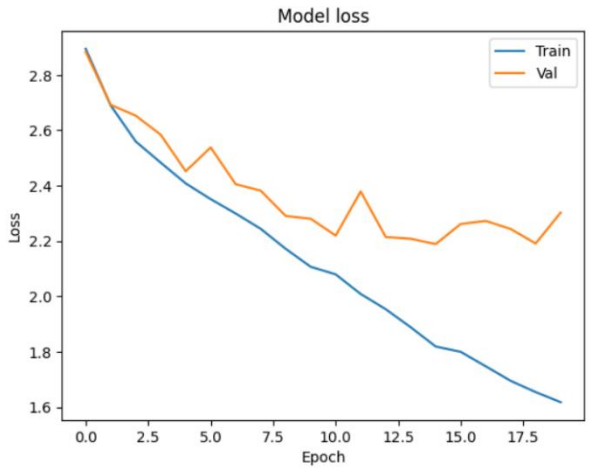
- **Precision, Recall, and F1-Score**:

- All metrics are very low (close to 0) for most classes, indicating that the model performs poorly in identifying the correct class and managing false positives and false negatives.
- The macro average precision, recall, and F1-score are extremely low (~0.02), showing no substantial performance improvement across the classes.

- **Overall Accuracy**:
  - The test accuracy is only **7%**, far below the expected performance, reflecting the model's inability to classify correctly on the testing dataset.

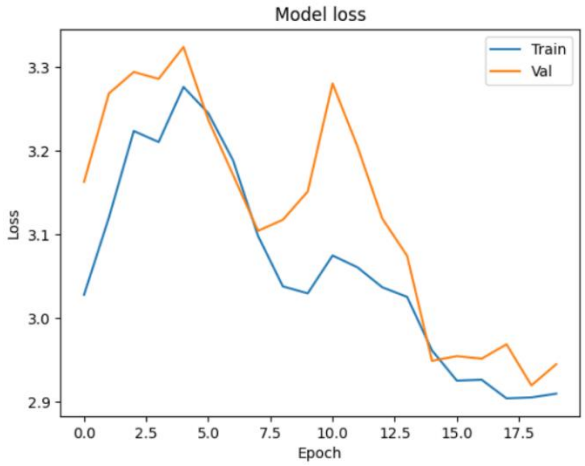
1.11

a. Learning Rate = 0.0001:



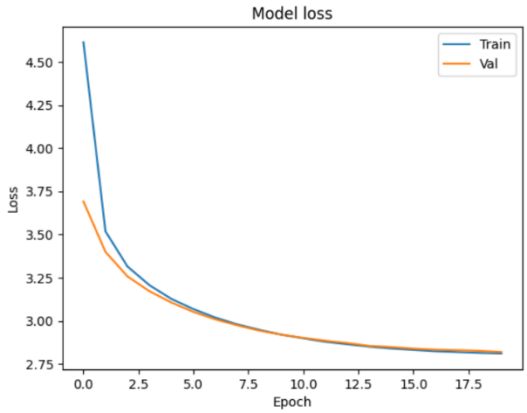
Both training and validation loss curves show a smooth and steady decrease, indicating consistent learning and generalization without overfitting.

b.Learning Rate = 0.001:



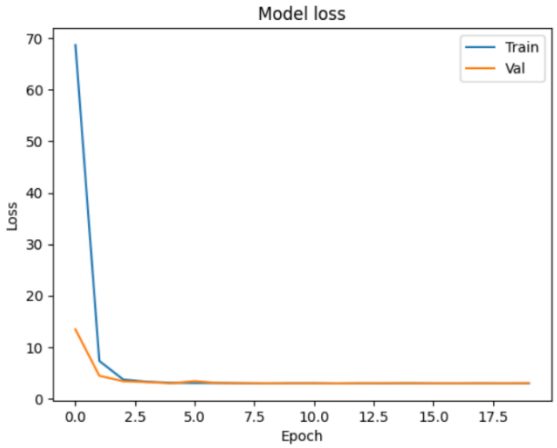
The loss decreases faster than for 0.0001, but validation loss has higher fluctuations, suggesting some degree of overfitting or learning instability.

c. Learning Rate = 0.01:



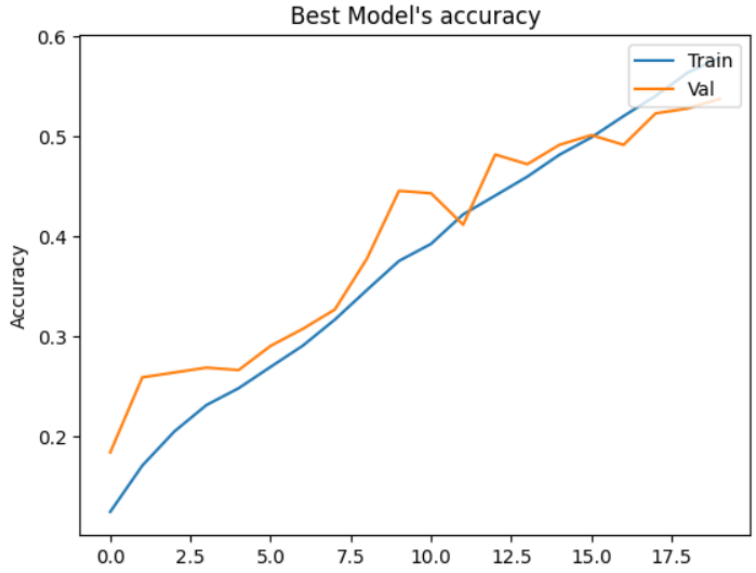
The training loss decreases rapidly but with significant fluctuations, and the validation loss shows high instability. This indicates that the model struggles to converge and generalize well.

d.Learning Rate = 0.1:



The training loss drops very quickly at the beginning but does not stabilize, and the validation loss barely decreases after the first few epochs. This suggests divergence due to an excessively high learning rate.

The best model selection:



The model trained with a learning rate of **0.0001** is the best choice. This learning rate ensures stable convergence and balanced performance on both training and validation sets, achieving training accuracy of 57.89% and validation accuracy of 53.75%. The steady decrease in loss without significant fluctuations demonstrates its capability to generalize well to unseen data, making it the optimal learning rate for this experiment.