I am choosing to use the image scenario in this project. The scenario is that botanists would like to separate crop seedlings from weeds effectively. This will result in better crop yields and better care for the environment. My research question is, “Can we produce a reliable convolutional neural network that successfully classifies crop seedlings from weeds?” The botanist's goal for my research is to solve the classification problem for seedlings using modern analysis techniques. The industry-relevant type of neural network I will use to solve this classification problem is a Convolutional Neural Network (CNN). The reason I am choosing a CNN is due to its efficient image processing, robust to the noise from image data, high accuracy, and automated feature extraction that recognizes the patterns (AspiringYouths, 2024).

For exploratory data analysis on the chosen image set, I have created a visualization of the distribution for the different classes and sample images with associated labels. I can see there is more of a few seedlings like loose silky-bent, scentless mayweed, common chickweed, and small-flowered cranesbill. I expect this will help the model predict these values better as they have the most data, but maybe it will have a harder time with seedlings like common wheat, shepherd's purse, maize, and black grass. The random sample images with labels show me some significant differences between some of these and others may appear more similarly. It will be interesting to see how this impacts CNN, as this will make predictions more difficult when there are some similarities. Lastly, there are only 4750 images, and there is not much data to help make this CNN.

A graph of a number of images

AI-generated content may be incorrect.A collage of images of plants

AI-generated content may be incorrect.

With the lack of data, I found it important to perform data augmentation to increase the dataset size. Augmentation will help increase the CNN robustness, help prevent overfitting and help prevent shortcuts in learning that may lead to a lower validation accuracy (Udacity, n.d.). I utilized a few different methods in my image augmentation. I used rotation range, width shift range, height hit range, shear range, zoom range, horizontal flip, and fill mode. This provided us with new train images that were created from current images, whether it was rotating them, changing height or width, shifting the image, zooming in on the image, flipping the image, and even fill mode to help fill empty places on the newly augmented data (Maximinusjoshus, 2022).

I encoded the labels and normalized the images. The encoding was straightforward in the process as we used label\_encoder in several projects prior to normalizing labels (*LabelEncoder*, n.d.). Now, to normalize the images, I used images = images / 255.0 as it would effectively cover the max intensity of colors (*How Do I Normalize the Pixel Value of an Image to 0~1?*, n.d.). As far as splitting my data into train-validation-test split, I first split the data into training and test data with an 80/20 ratio, so we could have plenty of test data and plenty of train data. I further split the train data 80/20 to have validation data. I knew I planned on augmenting the data, so the size of the training data was going to grow, and I would have plenty to support the size of the validation data.

<https://keras.io/api/layers/convolution_layers/convolution2d/>

There were different components of my CNN architecture. The number of nodes was 49,152, as the input layer shape was 128, 128, and 3 shapes (3 for the RGB images). The Conv2D layer uses 32 filters and 3x3 kernels as it covers the color depth, height and weight of the color image. As you can see from the summary screenshot below, I used 8 layers with convolutional, max pooling, flatten, dense, and dropout as the different types of layers. There were 29,517,260 total parameters (Udacity, n.d.). The activation functions were ReLU activation for the hidden layer (*ReLu and Dropout in CNN*, n.d.) and SoftMax activation for the output layer. SoftMax activation is used as it classifies the probability distribution, the output is easily interpretable, and considers all classes together (GeeksforGeeks, 2024). For max pooling I set the kernel size and stride to 2, so we could maintain more of the details with a smaller stride and window (Udacity, n.d.). I flattened the data and used a dense of 512 as this would allow for more datils to be gained (*Convolutional Neural Network (CNN)*, n.d.). The dropout layer at 0.5 would help with overfitting that I found originally and the ReLu I used in my convolutions was to help cover the non-linear relationships as well (*ReLu and Dropout in CNN*, n.d.). I did utilize the Adam optimizer at first as it is standard, and I felt like it was a great start (Brownlee, 2021).

A screenshot of a computer

AI-generated content may be incorrect.

From my original CNN, I decided to test backpropagation and hyperparameters in my optimized model. The loss function I utilized was the sparse categorical cross entropy as I used label encoding for my CNN (GeeksforGeeks, 2025). As I said before, I tried using the Adam optimizer, but it wasn’t giving me the results I wanted, as it was overfitting. I decided to try out Stochastic Gradient Descent (SGD) as it would be less susceptible to noise and would update the gradient with respect to the loss function (Van Otten, 2023). For my learning rate, I used the reduced learning rate as this could lower the learning rate when CNN no longer saw improvements. This would help improve CNN’s effectiveness (*ReduceLROnPlateau — PyTorch 2.6 Documentation*, n.d.). My stopping criteria were covered using the early stopping function to reduce training time, improve performance, and prevent overfitting. Monitoring the validation loss with patience of 5 means that after 5 epochs with no improvements to validation loss, early stopping occurs. This early stopping leads to reduced training time, prevents overfitting and improves performance (Bhattbhatt, 2024).

A screenshot of a graph

AI-generated content may be incorrect.

The correlation matrix showed me where my CNN was classifying accurately and where it wasn’t classified accurately. While most of the results were decent, I did notice the CNN was having trouble with loose silky-bent. It classified black-grass accurately 10 times, but 28 times it incorrectly identified it as loose silky-belly. This problem could be seen again with common wheat only being accurately classified 7 times, with the CNN inaccurately classifying it as loose silky-bent 17 times. When I first looked at my data I noticed plenty of images of the loose silky-bent, but both black-grass and common wheat had a very small sample size. I would suggest possibly improving this CNN by obtaining more of the black-grass and common wheat. I could also try to augment more of those in the next CNN.

A screenshot of a data

AI-generated content may be incorrect.

A number of numbers on a white background

AI-generated content may be incorrect.

The model training is provided above. The first epoch started with 12% accuracy and 20% validation accuracy, but over the 31 epochs, that increased to 90.72% accuracy and 70% validation accuracy. As noted above, this was partially expected when looking at our confusion matrix and the troubles it had with some of the classifications. I had set the epoch to 50, but as we can see above the early stopping led to only 31 out of 50 epochs. This ensured the model didn’t overfit, as we can see that around the 70% validation accuracy the model started to stagnate on increasing the performance. We can further see the comparison of the training data to the validation dataset becoming better as our loss function started at 2.447 and decreased all the way down to 0.2944. This decrease demonstrates that the iterations of the loss function were able to decrease in our CNN, as it was adjusting accurately over time. CNN increased the proficiency of the training data by classifying the images accurately according to the validation set. As mentioned before, to address the overfitting, I used the early stop function; I tried the SGD optimizer instead of the Adam optimizer and used data augmentation to produce plenty of training data. The predictive accuracy appears to be at about 70% after validation.

A graph of training and validation loss

AI-generated content may be incorrect.



My neural network is functional. The network architecture improved through optimization, as well as from the architecture. SGD’s advantages were obvious with its accuracy and its ability to tune both the learning rate and momentum (*Stochastic gradient descent*, n.d.). Utilizing the learning rate reduction and early stopping helped ensure the best learning rate was being used over time with early stopping reducing overfitting (*ReduceLROnPlateau — PyTorch 2.6 Documentation*, n.d.). The validation accuracy is only at about 70% with an accuracy of about 90%. This does demonstrate overfitting, and I do suggest more work be done on this model. I did answer the question of whether I can produce a reliable CNN that can successfully classify seedlings from weeds. Their goal of having their classification problem solved by modern analysis techniques is made, but it is at only about 70%. I learned lessons that improved my CNN. This ranged from data augmentation to provide more data, optimizers like Adam and SGD, and the early stopping to function to reduce overfitting. I recommend further action to increase accuracy. This further action can range from collecting more data on black-grass and common weed, providing more augmented data on the two, or even looking to find other datasets that may align with this project.

References:

AspiringYouths. (2024, February 19). *Advantages and Disadvantages of Convolutional Neural Network (CNN)*. <https://aspiringyouths.com/advantages-disadvantages/convolutional-neural-network-cnn/#google_vignette>

Bhattbhatt, V. (2024, January 10*). A Step-by-Step Guide to Early Stopping in TensorFlow and PyTorch*. Medium. <https://medium.com/@vrunda.bhattbhatt/a-step-by-step-guide-to-early-stopping-in-tensorflow-and-pytorch-59c1e3d0e376>

Brownlee, Jason. (2021, January 13th). *Gentle Introduction to the Adam Optimization Algorithm for Deep Learning*. Deep Learning Mastery. <https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/>

Code With Prince. (2022, June 20). *Image classification using CNN (CIFAR10 dataset) | Deep Learning Tutorial (Tensorflow & Python)* [Video]. YouTube. <https://www.youtube.com/watch?v=zadvMgHaTfA>

*Convolutional Neural Network (CNN)*. (n.d.). TensorFlow. <https://www.tensorflow.org/tutorials/images/cnn>

GeeksforGeeks. (2024, November 19). *Softmax activation function in neural networks*. GeeksforGeeks. <https://www.geeksforgeeks.org/the-role-of-softmax-in-neural-networks-detailed-explanation-and-applications/>

GeeksforGeeks. (2025, January 9). *Sparse Categorical Crossentropy vs. Categorical Crossentropy*. GeeksforGeeks. <https://www.geeksforgeeks.org/sparse-categorical-crossentropy-vs-categorical-crossentropy/>

*Keras ImageDataGenerator: random transform*. (n.d.). Stack Overflow. <https://stackoverflow.com/questions/47461538/keras-imagedatagenerator-random-transform>

*Keras Early Stopping: Monitor “loss” or “val\_loss”?* (n.d.). Data Science Stack Exchange. <https://datascience.stackexchange.com/questions/73726/keras-early-stopping-monitor-loss-or-val-loss>

*How do I normalize the pixel value of an image to 0~1?* (n.d.). Stack Overflow. <http://stackoverflow.com/questions/70783357/how-do-i-normalize-the-pixel-value-of-an-image-to-01>

*How to save final model using keras?* (n.d.). Stack Overflow. <https://stackoverflow.com/questions/42763094/how-to-save-final-model-using-keras>

Ifttt-User. (2023, July 25). *Keras EarlyStopping Callback to train the Neural Networks Perfectly*. Towards AI. <https://towardsai.net/p/l/keras-earlystopping-callback-to-train-the-neural-networks-perfectly>

*LabelEncoder.* (n.d.). Scikit-learn. <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html>

Maximinusjoshus. (2022, January 7*). Image Data Augmentation using Keras ImageDataGenerator.* Medium. <https://medium.com/featurepreneur/image-data-augmentation-using-keras-imagedatagenerator-1cee60255ea8>

*ReLu and Dropout in CNN*. (n.d.). Stack Overflow. <https://stackoverflow.com/questions/27319931/relu-and-dropout-in-cnn>

*ReduceLROnPlateau — PyTorch 2.6 documentation*. (n.d.). <https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.ReduceLROnPlateau.html>

*Scribbr.* (n.d.). Scribbr. <https://www.scribbr.com/>

Soni, P. (2023, November 8). *CNN model with PyTorch for image classification - TheCyPhy - Medium*. Medium. <https://medium.com/thecyphy/train-cnn-model-with-pytorch-21dafb918f48>

*Stochastic gradient descent.* (n.d.). Scikit-learn. <https://scikit-learn.org/stable/modules/sgd.html>

Udacity. (n.d.). *Convolutional neural networks*. <https://learn.udacity.com/paid-courses/cd1821>

Van Otten, N. (2023, October 31). *Adam Optimizer explained & how to use in Python [Keras, PyTorch & TensorFlow]*. Spot Intelligence. <https://spotintelligence.com/2023/03/01/adam-optimizer/>

*Which parameters should be used for early stopping?* (n.d.). Stack Overflow. <https://stackoverflow.com/questions/43906048/which-parameters-should-be-used-for-early-stopping>