The aim of this report is to describe the approach taken to solve a classification problem using machine learning techniques on a dataset containing information on E-commerce products. The dataset is provided in three formats - categorical attributes, noisy text descriptions, and noisy images of the products. The goal is to accurately classify the agricultural products into one of 27 possible classes based on the given information. To accomplish this task, a combination of natural language processing, deep learning, and ensemble learning techniques are used.

**Categorical Attributes Data**

Given that the categorical attributes were provided in string format, some work needed to be done before it could be used for predictions. To process this, the string data is transformed into a vector of categorical variables that have been encoded using the LabelEncoder from scikit-learn. Each categorical attribute is first transformed into a numerical value using the LabelEncoder, which assigns a unique integer value to each category. The tensor of encoded values is then used as input for the model.

To make predictions using categorical attributes, a deep neural network (DNN) is used. The DNN is a type of artificial neural network that consists of multiple layers of interconnected nodes or "neurons". The DNN has three fully connected layers, each followed by a rectified linear unit (ReLU) activation function. The final layer uses a softmax activation function to output the predicted probability distribution over the 27 possible classes. To prevent overfitting, a dropout layer with a dropout rate of 0.25 is included between each fully connected layer. The DNN is trained using the cross-entropy loss function and the Adam optimizer with a learning rate of 0.001.

**Noisy Text Data**

To handle the noisy text, a combination of natural language processing and deep learning techniques was used to make predictions based on noisy text descriptions. The noisy text is first processed using tokenization, stopword removal, and part-of-speech tagging to extract relevant keywords. To be more specific, in yield\_tokens and process\_text, the input noisy text is first denoised by removing the non-alphabetical characters and single character tokens, as well as adding spaces before capital letters to break up any continuous text. The text data is then tokenized and tagged (with its parts-of-speech or POS tag) using the functions from the nltk library. It then extracts all keywords from the tagged tokens by selecting only those tokens that have a POS tag starting with 'NN' (nouns). A vocabulary object is created from the tokens yielded by the generator described above to encode the text data, which is then fed into a neural network model.

The class TextModel is the neural network model I created for text classification, consisting of an embedding layer followed by a fully connected layer to predict the class label of the input text. The embedding layer maps each token in the input text to a dense vector which is then fed into the linear layer to produce class predictions. The model is trained using backpropagation and the Adam optimizer, with a cross-entropy loss function to optimize the model parameters for accurate predictions.

**Noisy Image Data**

For the images, there were several techniques combined to make the predictions. To start off, to reduce noise in the provided noisy images, a Gaussian filter is applied to each image. A Gaussian filter is a type of linear filter that is commonly used for smoothing images by convolving the image with a Gaussian kernel, with the level of smoothing controlled by the sigma value. Since the sigma value is set to 1.25 (picked after some trial and error with balancing the denoising effect with the blurriness), the filter will apply a moderate amount of smoothing to the image to remove noise and small details from the image while preserving larger-scale features, at the cost of losing image sharpness and detail.

Furthermore, a transform is applied to the images to further denoise the images and to improve the stability and convergence of the model during training. For images, it is important to normalize the pixel values so that they are centered around zero and have a unit variance, which makes it easier for the model to learn meaningful patterns in the data. To be more specific, after applying transform to an image, the pixel values of the image would be normalized to have a mean of zero and a standard deviation of one. This the pixel intensity values, but not the visual content of the image. Overall, this normalization step can help to improve the accuracy and stability of the machine learning model by reducing the impact of variations in pixel intensity across different images in the dataset.

The following shows the preprocessing of a sample image:

A picture containing text, mammal

Description automatically generated A close up of a person

Description automatically generated with low confidence A close up of a person's face

Description automatically generated with low confidence

The above two techniques transform the input images into a format that is suitable for input to a model, and in hopes of improving the accuracy and stability of the model. In terms of the model architecture, I used a CNN with two stacks of (Conv2d, Activation, Max pooling) layers connected to a dense neural net, with dropout probability of 0.25 after each max pooling and fully connected layer. I chose a CNN because it is designed to handle image data, with convolutional and pooling operations that extract local features from the image, pooling layers that reduce the spatial dimensionality of the feature maps (while retaining the most important information), and fully connected layers at the end of the CNN map the high-level features extracted from the convolutional layers to the output classes. In terms of the finer details, ReLU was used as the activation function to avoid the vanishing gradient problem, Adam was chosen as the optimizer for its faster convergence and better generalization from momentum, and dropout was implemented to avoid overfitting and improve generalization.

**Ensemble Learning**

In order to combine the models, a voting function is implemented and used similar to the bagging technique. In bagging, each base model is trained independently of the others, using a random subset of the training data, and the final prediction is made by aggregating the predictions of all base models. This can help reduce the variance of the model and prevent overfitting. Bagging is often used with decision trees, where each base model is a different tree trained on a random subset of the data. The final prediction is then made by averaging the predictions of all the trees.

However, the dataset given is slightly different from this typical bagging use case since each data entry has different features, and instead of training models on random subsets, they each specialize in handling a specific part of the data. In the above voting algorithm, the three models (CNN, DNN, and text model) are trained on the same data but with different types of features, and their predictions are combined using a simple voting mechanism. The final prediction is the one that receives the most votes. This approach was used with the goal of helping to capture different aspects of the data and to improve the overall performance of the model.