Results (for reference)

Decision tree

Best Parameters found: {'criterion': 'entropy', 'max_depth': 40, 'max_features': 'sqrt', 'min_samples_leaf': 18, 'min_samples_split': 33}
Best Cross-Validation Accuracy: 0.2301
Final Test Set Accuracy: 0.2528

Ensemble methods - Random Forest

Best Parameters found: {'max_depth': 20, 'max_features': 'sqrt', 'min_samples_leaf': 4, 'min_samples_split': 34, 'n_estimators': 399}
Best Cross-Validation Accuracy: 0.4650
Final Test Set Accuracy: 0.4714

Support Vector Machine

Best Parameters found: {'C': np.float64(621.381353074258), 'gamma': np.float64(0.013847288133351985), 'kernel': 'rbf'}
Best Cross-Validation Accuracy: 0.5394
Final Test Set Accuracy: 0.5637

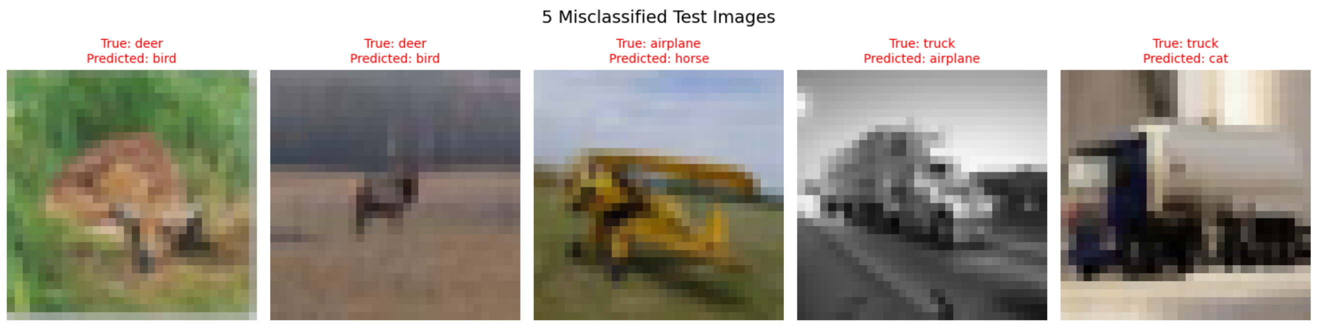
Report Task 3

For the decision tree, I used RandomizedSearchCV for the cross-validation with n_iter=100. I could do this because decision trees are much cheaper to train than ensembles and support vector machines. As such, I was able to explore the hyperparameter space more thoroughly.

For the ensemble methods, I chose random forest as an improvement on the single decision tree, as it introduces bagging. Cross-validation was performed with n_iter=50, since random forest classifiers take much longer to train. I chose the upper limit of n_estimators to be relatively high, as adding more trees does generally not cause overfitting, but stabilises the error. I didn't test on lower values because they result in high variance

For the support vector machine, I performed cross-validation on log (instead of uniform) distributions of the hyperparameters. This was so I could explore, for example, 0.1, 1, 10, and 100 with equal probability, instead of spending 90% of my time searching between 100 and 1000. Initially I tried using n_iter=50, but due to the computational cost of support vector machines, I had to reduce it to 20.

For all the models, I first tried using GridSearchCV. I abandoned this because checking every combination of the defined grids took an extremely long amount of time, and did not allow me to use as wide a range of hyperparameter values and train on as many hyperparameters as I wanted to. Instead, I switched to using RandomSearchCV, which allowed me to find better models in less time by exploring the space more efficiently.



The misclassified images above are from the decision tree model. Decision trees struggle to detect edges or complex forms, and so it was likely that mine made splits based off of the dominant background colour.
For example, image 3 shows a brown/yellow airplane on a green background. Horses are usually on green grass, and airplanes are usually on blue sky, and so the model likely saw the green background and guessed horse.

We can also see this lack of ability to distinguish edges in the 5th image. Airplanes are often grey/white objects against a light background, but since the tree cannot see that the object has wheels or a cab, it simply sees "light grey pixels" and classifies it as the most common grey object: an airplane.

Report Task 4

To reiterate, the accuracy on the test set for each of the models was as follows:

- Decision Tree: 25.28%
- Random Forest: 47.14%
- Support Vector Machine: 56.37%

Focussing first on the decision tree, we can see it has the lowest accuracy. It likely performed poorly because image data is inherently complex, however a single tree attempts to classify images using simple, axis-aligned splits, and this fails to capture the spatial relationships required for image recognition. Single decision trees have very high variance, and so small changes in image noise can lead to completely different tree structures. The low accuracy suggests the model is too simple to map the high-dimensional pixel space to class labels effectively.

For the random forest classifier, there is a jump in accuracy. This shows how, through bagging, the random forest smoothed out the noise and variance inherent in the single decision tree. By considering only a subset of pixels at each split across 399 trees, the model ensured that dominant (but perhaps noisy) background pixels don't control every decision, allowing the model to learn a more robust generalised pattern.

The support vector machine is clearly the best performer, and is much better suited for high-dimensional feature spaces (such as pixel data). Instead of making crude splits, it searches for an optimal hyperplane that maximizes the margin between classes. The Radial Basis Function kernel used maps the data into an high-dimensional space where non-linear patterns become linearly separable, allowing the support vector machine to model complex decision boundaries.

Despite the support vector machine being the best performing, the time complexity is at best $O(N^2)$, and so I explored less combinations of hyperparameters than with the other models. Perhaps, with longer training times or faster computers, I could have found a more optimal set of hyperparameters. The decision tree is by far the cheapest and fastest to train, and explored many more hyperparameter combinations, but clearly support vector machines are so much more suited to high-dimensional image data that this did not make a difference.