Enhancing Deep Learning Image Classification for Robotics Applications

Introduction

Image classification stands as a cornerstone in the realm of robotics, providing robots with the crucial ability to interpret and interact with the visual world around them [General knowledge]. Accurate recognition of objects within their environment is paramount for robots to perform a wide array of tasks, including autonomous navigation, precise object manipulation, and seamless human-robot interaction [General knowledge]. In recent years, the field of deep learning has emerged as a transformative force in image classification, achieving unprecedented levels of performance on intricate and challenging tasks [General knowledge].

The user in this scenario has developed a foundational deep learning image classification model utilizing the TensorFlow and Keras libraries. This model is designed to classify images from the CIFAR-10 dataset and employs a sequential Convolutional Neural Network (CNN) architecture [User Query]. The ultimate goal is to integrate this image classification capability into the main control system of a robotics project [User Query]. To achieve this integration effectively, the user seeks guidance on several key aspects, including improving the prediction accuracy of the model and restructuring the existing code into a more organized and reusable format as a Python class [User Query].

This report aims to provide expert-level guidance to the user to meet these objectives. It will delve into best practices for structuring the TensorFlow/Keras code as a Python class, which will enhance the maintainability and reusability of the image classification module.[1] Furthermore, the report will explore and recommend more advanced CNN architectures specifically known for their strong performance on the CIFAR-10 dataset, taking into account both their accuracy and computational complexity.[7] Techniques for significantly improving the accuracy of the image classification model, such as data augmentation and normalization, will be thoroughly detailed.[16] The report will also investigate strategies for hyperparameter tuning, a critical process for optimizing the performance of deep learning models.[41] Finally, the report will cover the essential procedures for properly saving and loading the trained model and for preprocessing new images to ensure accurate predictions within the robotics application.[68]

The user's current endeavor signifies a common progression in machine learning projects, transitioning from an initial experimental script to a more robust and fully

integrated solution. The emphasis on achieving exact prediction accuracy for a robotics project underscores the vital role of reliable perception in such applications.

Best Practices for Structuring TensorFlow/Keras Code as a Python Class

Adopting an object-oriented programming approach through the use of Python classes offers several advantages for structuring deep learning projects.[3] One key benefit is the enhanced modularity achieved by encapsulating different components of the model, such as the layers, training procedures, and prediction steps, into reusable units.[3] This modularity leads to improved organization of the codebase, making it easier to manage and maintain over time.[3] Once a class is defined for the image classification model, it can be easily instantiated and utilized in various parts of the robotics project as needed.[3] This reusability prevents code duplication and promotes a more efficient development process.[6] The structure provided by classes makes the entire codebase more understandable and easier to navigate, which is particularly beneficial for larger projects or when multiple developers are involved.[3] Furthermore, a class can effectively manage the state of the trained model, including its learned weights and architectural configuration, along with other relevant parameters like class names and the expected input image size, as attributes of the class instance.[6] Utilizing instances and instance methods is particularly advantageous when the state of the model or related parameters needs to be modified during the application's lifecycle.[6]

When designing a robust class structure for the image classification model, a typical approach involves including methods for several key functionalities. These commonly include an initialization method to set up the model, a method to define and build the model architecture, a training method to handle the learning process, methods for saving and loading the trained model, and a prediction method to classify new images. For enhanced flexibility, it is often beneficial to separate the definition of the model's architecture from the logic for training and making predictions.[3] This separation allows for easier modification of the model's structure or the training procedure without affecting other parts of the class.[3] The class should also ideally handle the loading and preprocessing of image data internally to ensure consistency across the application. This can be achieved through dedicated methods within the class, such as static methods for loading data from specific sources.[6]

The implementation of several key methods within the class is crucial for its functionality. The __init__(self,...) method, also known as the constructor, should be responsible for initializing essential parameters required for the model. These

parameters might include the expected input image shape (e.g., 32x32x3 for CIFAR-10), the number of classes to be predicted (e.g., 10 for CIFAR-10), and potentially any hyperparameters that will be used during training.[6] This method can also be used to load the names of the classes from a predefined list or file.[6] The build_model(self) method will contain the definition of the CNN architecture using the layers provided by TensorFlow and Keras.[1] Keras offers different APIs for building models, including the Sequential API for linear stacks of layers, the Functional API for more complex architectures with multiple inputs or outputs, and the Subclassing API for complete customization.[1] The choice of API will depend on the complexity of the CNN architecture that is ultimately selected. This method should typically return the compiled Keras model. The train(self, training_images, training_labels,...) method will encapsulate the model training process.[1] This includes compiling the model using a specified optimizer (e.g., Adam), a suitable loss function (e.g., SparseCategoricalCrossentropy for integer labels), and relevant metrics (e.g., SparseCategoricalAccuracy). Subsequently, the method will call the fit() method of the compiled model to train it on the provided training images and labels for a specified number of epochs and batch size. The method should also handle the inclusion of a validation set if needed to monitor the model's performance during training. The save_model(self, filepath) method will be responsible for saving the trained model to a specified file path.[71] Keras provides the save_model() function for this purpose, which can save the entire model, including its architecture, weights, and optimizer state.[69] The load_model(self, filepath) method will load a pre-trained model from a given file path.[71] The tf.keras.models.load_model() function is used to load models that were previously saved using save_model().[69] Finally, the predict(self, image) method will take a preprocessed image as input and return the model's prediction.[83] This method will likely involve expanding the dimensions of the input image to match the model's expected input shape (including the batch dimension) and ensuring that the image is normalized correctly before passing it to the model's predict() method. For multi-class classification, the np.argmax() function is typically used on the output of the predict() method to obtain the index of the class with the highest probability.[92]

To make the image classification functionality easily executable, it is beneficial to incorporate a main function into the script. This function will serve as the primary entry point when the script is run. Within the main function, the necessary steps for using the image classification model should be orchestrated. This might include loading the training and testing data, instantiating the image classification model class, training the model (if a pre-trained model is not being used), loading a pre-trained model from a file, loading a new image that needs to be classified,

preprocessing this image to the format expected by the model, and then calling the predict() method to obtain the classification result [User Query]. This structure with a dedicated main function allows for a clear and organized way to run and demonstrate the image classification capabilities.

Structuring the code in this manner, using a Python class, offers a more organized and maintainable approach to the project. The separation of concerns into distinct methods for model building, training, and prediction enhances the reusability of the code and makes it easier to modify individual components without causing unintended side effects in other parts of the system. The inclusion of a main function provides a well-defined starting point for the execution of the image classification pipeline. The choice of the specific Keras API (Sequential, Functional, or Subclassing) for the build_model method should be carefully considered based on the complexity of the CNN architecture that will be chosen in the subsequent steps. For simpler architectures like the user's initial model, the Sequential API might suffice, while more advanced architectures might necessitate the use of the Functional or Subclassing APIs.

Exploring Advanced Convolutional Neural Network Architectures for CIFAR-10

The user's current image classification model employs a sequential CNN architecture, which is a fundamental approach to image classification [User Query]. This type of model, consisting of convolutional layers, pooling layers, and fully connected layers, can be effective for initial experimentation. However, to achieve higher levels of accuracy, especially on a challenging dataset like CIFAR-10, exploring more advanced CNN architectures is often necessary [General knowledge]. Even a relatively simple CNN like LeNet can achieve a reasonable accuracy (around 76%) on CIFAR-10 with the application of appropriate training techniques.[7]

Several advanced CNN architectures have demonstrated superior performance on the CIFAR-10 dataset. Among these, **ResNet (Residual Network)** stands out as a particularly effective option.[7] ResNet architectures are characterized by the introduction of "skip connections" or "residual blocks".[7] These connections allow the network to learn residual mappings, which simplifies the training of very deep networks by helping to mitigate the vanishing gradient problem.[7] The core idea is that instead of directly learning a complex mapping, the network learns the residual (the difference between the input and the desired output), which is often easier to optimize. Various ResNet variants have been developed, including those specifically designed for the CIFAR-10 dataset, such as ResNet-20, ResNet-32, and ResNet-110.[7]

These variants differ primarily in the number of layers and residual blocks they contain, affecting their complexity and representational capacity. The use of skip connections is a key feature that enables the training of much deeper networks compared to traditional sequential models, leading to improved accuracy.[7] ResNet architectures for CIFAR-10 can vary in complexity, with smaller versions like ResNet-20 having fewer parameters (around 0.27M to 0.47M) compared to deeper ones like ResNet-110 (around 1.7M).[7] ResNet models have achieved high accuracy on CIFAR-10, with reported error rates as low as 2-4% when combined with data augmentation.[7] For instance, one study reported achieving 94% accuracy on CIFAR-10 using a relatively simple ResNet architecture within a short training duration.[95] However, it is important to note that achieving the highest levels of accuracy often requires careful hyperparameter tuning and the use of effective data augmentation techniques.[93] Some users have reported lower accuracy with deeper ResNet variants like ResNet-110, highlighting the importance of proper configuration.[93]

Another popular CNN architecture is **VGG (Visual Geometry Group)**.[7] VGG networks are known for their deep structure, which is constructed using a stack of convolutional layers with very small 3x3 filters, followed by max-pooling layers.[7] The uniformity in the architecture, with the consistent use of 3x3 filters throughout the network, is a defining characteristic of VGG.[98] Stacking multiple 3x3 convolutional layers can achieve the same receptive field as a single larger kernel but with the benefits of increased non-linearities and a reduction in the number of parameters. Common VGG variants include VGG16 and VGG19, which differ in the total number of layers.[7] However, VGG models, particularly the deeper variants, can be computationally expensive due to their large number of parameters.[7] For example, VGG19 has approximately 39 million parameters.[7] While VGG can achieve good accuracy on CIFAR-10, with some studies reporting around 91-93% accuracy when combined with data augmentation, it may not be as parameter-efficient or achieve the same state-of-the-art results as ResNet or more modern architectures for this specific dataset.[7] Some reports indicate that VGG models might yield lower accuracy on CIFAR-10 if they are not properly tuned for the dataset's smaller image size (32x32 pixels).[100] Furthermore, directly applying transfer learning from VGG models pre-trained on the much larger ImageNet dataset might not be directly optimal for CIFAR-10 without careful fine-tuning of the architecture to suit the smaller input size.[98] Users have also reported encountering issues with low validation accuracy when attempting to train VGG19 on the CIFAR-10 dataset.[100]

In addition to ResNet and VGG, several other advanced CNN architectures are relevant for image classification on CIFAR-10. **DenseNet (Densely Connected**

Convolutional Networks) is an architecture where each layer is connected to every other layer in a feed-forward manner.[7] This dense connectivity promotes the reuse of features across the network and helps in propagating gradients more effectively, making the network more parameter-efficient.[7] DenseNets have demonstrated excellent performance on the CIFAR-10 dataset.[7] **EfficientNet** is another architecture that has achieved state-of-the-art accuracy with a significantly lower number of parameters compared to other high-performing models.[97] It achieves this by scaling all dimensions of the network—depth, width, and input resolution—in a principled and balanced way.[97] **Network in Network (NIN)** introduces the concept of using micro-networks, such as multilayer perceptrons, within the convolutional layers to enhance the model's representational power.[7] **ResNeXt** is an architecture that improves upon the ResNet structure by using aggregated residual transformations, employing a "cardinality" dimension to enhance the residual blocks.[7]

Based on the performance and efficiency considerations for a robotics application, **ResNet** appears to be a strong candidate for achieving higher accuracy on CIFAR-10.[7] Its effectiveness in training deep networks through the use of skip connections has been well-established. For a robotics platform where computational resources might be a concern, starting with a smaller ResNet variant, such as ResNet-20 or ResNet-32, could be a good approach. **DenseNet** is another excellent option that offers high accuracy along with parameter efficiency, making it suitable for resource-constrained environments as well.[7] While **EfficientNet** has achieved very high accuracy, its architecture might be more complex to implement from scratch, and the very high accuracy reported often comes with a significant increase in the number of parameters, which might not be ideal for all robotics applications.[97] **VGG**, while capable of achieving reasonable accuracy, tends to be more computationally intensive than ResNet or DenseNet for comparable or slightly lower performance on CIFAR-10.[7] Its deeper structure and larger number of parameters can also make it more susceptible to overfitting on smaller datasets like CIFAR-10.[98]

The following table provides a comparison of these CNN architectures on the CIFAR-10 dataset:

Architecture	Approximate Top-1 Accuracy (with Augmentation)	Approximate Parameter Count	Relative Complexity	Key Features
User's Model	Potentially lower (to be	Lower	Lower	Sequential

	determined)			layers
ResNet (e.g., 20)	Around 91-93%	~0.27M-0.47M	Moderate	Residual blocks, skip connections
ResNet (e.g., 110)	Around 93-94%	~1.7M	Higher	Deeper version of ResNet
VGG19	Around 93-94%	~39M	Very High	Deep with small convolutional filters
DenseNet-100	Around 94-95%	~0.85M-3.3M	Moderate to High	Dense connections, feature reuse
EfficientNet-B7	Around 98-99%	~66M	Very High	Scaled depth, width, and resolution

This table offers a concise overview of the trade-offs between different CNN architectures in terms of accuracy, model size (indicated by parameter count), relative complexity, and their defining architectural features. This information should assist the user in making an informed decision based on the specific requirements and resource constraints of their robotics application.

Enhancing Accuracy through Data Augmentation and Normalization Techniques

To further enhance the accuracy of the image classification model, especially when working with a dataset of limited size like CIFAR-10, employing data augmentation and appropriate normalization techniques is crucial.[16]

Data augmentation is a powerful strategy that artificially expands the size of the training dataset by applying a variety of transformations to the existing images.[16] This process helps the model to generalize better to new, unseen data and significantly reduces the risk of overfitting, particularly when the original dataset is relatively small.[16] By exposing the model to different variations of the same object, such as images that are rotated, flipped, or zoomed, the model learns to extract more robust and invariant features that are not specific to a particular orientation or viewpoint.[16]

TensorFlow provides a convenient tool for implementing data augmentation called ImageDataGenerator.[8] This utility allows for the easy application of a wide range of image transformations. Common augmentation techniques that are particularly suitable for the CIFAR-10 dataset include random rotations (by a small angle, e.g., 10 to 20 degrees) [16], which helps the model become invariant to slight changes in object orientation. Random horizontal and vertical flips [16] can also be effective, although for CIFAR-10, horizontal flipping is generally more applicable than vertical flipping for most classes.[29] Random width and height shifts (by a fraction of the image dimensions, e.g., 0.1) [8] can help the model learn to recognize objects even if they are not perfectly centered in the image. Random zooming (e.g., by a factor of 0.1 to 0.2) [16] can further improve generalization. Other techniques like random cropping [25], shear transformations [29], and adjustments to brightness and contrast [16] can also be beneficial. The flow() method of the ImageDataGenerator can be used to generate batches of augmented images on the fly during the training process.[8] It is important to carefully select the augmentation techniques and their parameters to ensure they are relevant to the task and do not introduce unrealistic or harmful distortions to the images.[22] For instance, excessive augmentation could lead to images that no longer represent the original objects accurately.[16]

Image normalization is another critical step in enhancing the accuracy and training efficiency of deep learning models.[33] Normalization involves scaling the pixel values of the images to a specific range, typically between 0 and 1 or -1 and 1.[33] For the CIFAR-10 dataset, a common and effective practice is to normalize the pixel values to the range of 0 to 1 by dividing each pixel value by 255.0, as the initial pixel values are in the range of 0 to 255.[8] This normalization process ensures that all pixel values are on a similar scale, which helps the model to converge faster during training and can lead to an overall improvement in performance.[33] It also prevents features (in this case, pixel values) with larger ranges from dominating the learning process.[34] In addition to this basic pixel-wise normalization, more advanced normalization techniques, such as Batch Normalization, can be incorporated directly into the CNN architecture.[8] Batch Normalization normalizes the activations of each layer within each mini-batch during training.[35] This technique has been shown to further stabilize the training process, accelerate convergence, and improve the generalization ability of the model.[35] When applying normalization techniques to the CIFAR-10 dataset, it is essential to apply the same normalization method (e.g., dividing by 255.0) to both the training and testing images to maintain consistency.[8] Furthermore, considering the addition of Batch Normalization layers after the convolutional layers in the CNN architecture could lead to significant improvements in accuracy.[8]

In summary, data augmentation and normalization are critical techniques for enhancing the accuracy of deep learning image classification models, especially for datasets like CIFAR-10. Data augmentation helps to improve the model's ability to generalize by creating a more diverse training set, while normalization ensures that the input data is in a suitable range for efficient and stable training. The combination of these techniques is a fundamental step towards achieving high performance in image classification tasks.

Investigating Hyperparameter Tuning Strategies for Optimal Model Performance

Hyperparameter tuning is a crucial step in the process of developing high-performing deep learning models.[41] Hyperparameters are the settings that are configured before the training process begins and control various aspects of the model's learning.[42] Finding the optimal values for these hyperparameters can significantly impact the final performance and accuracy of the model.[41]

Several key hyperparameters play a significant role in the training of deep learning models. The **learning rate** determines the size of the steps taken to update the model's weights during training. A learning rate that is too high can cause the training process to diverge, while a learning rate that is too low can result in very slow convergence to a good solution.[42] The **batch size** specifies the number of training samples that are processed together before the model's weights are updated. The batch size can affect the speed of training and the generalization ability of the model.[61] The **number of epochs** defines how many times the entire training dataset is passed through the model during training.[51] Choosing an appropriate number of epochs is important to avoid underfitting (too few epochs) or overfitting (too many epochs). **Early stopping**, a technique to monitor the model's performance on a validation set and stop training when it starts to degrade, can be used to mitigate overfitting.[8] The choice of the **optimizer** (e.g., Adam, SGD, RMSprop, Adadelta) also plays a critical role, as different optimizers use different strategies to update the model's weights and can affect the training dynamics and the final performance achieved.[44] Beyond these, other hyperparameters related to the model's architecture, such as the number of filters in convolutional layers, the size of the convolutional kernels, the number of neurons in dense layers, the dropout rate used for regularization, and other regularization parameters, can also be tuned to improve performance.[44]

Various methodologies can be employed to find the best set of hyperparameters. **Manual tuning** involves trying different hyperparameter values based on experience,

intuition, and sometimes trial and error.[41] While this approach can be useful, it is often time-consuming and may not effectively explore the vast hyperparameter space.[41] **Automated hyperparameter tuning** methods provide more systematic and efficient ways to search for optimal hyperparameters. **Grid search** is an exhaustive search method that tries every possible combination of hyperparameters from a predefined set of values.[41] While thorough, grid search can become computationally very expensive, especially when dealing with a large number of hyperparameters or a wide range of possible values.[41] **Random search** offers a more efficient alternative by randomly sampling hyperparameter combinations from a defined range.[41] This method is often more effective than grid search, especially when only a subset of the hyperparameters significantly impacts the model's performance.[41] **Bayesian optimization** is a more advanced technique that uses a probabilistic model to guide the search for the best hyperparameters.[41] It considers the results of past evaluations to intelligently select the next set of hyperparameters to try, making it more efficient than grid or random search, particularly for computationally expensive models.[41] **Keras Tuner** is a powerful and user-friendly library in TensorFlow/Keras that implements Bayesian optimization, as well as other hyperparameter tuning algorithms like RandomSearch and Hyperband.[1] Hyperband is an efficient tuning algorithm that adaptively allocates more resources to promising hyperparameter configurations and uses early stopping to discard poorly performing ones, making the search process faster.[53]

For a robotics project where achieving high accuracy is a critical requirement, employing automated hyperparameter tuning is strongly recommended.[41] **Keras Tuner** provides an excellent framework for this purpose.[1] It simplifies the process of defining the search space for the hyperparameters of interest and efficiently finds the optimal combination that maximizes the model's performance.[54] The process typically involves defining a model-building function that includes the hyperparameters to be tuned, specifying the range of possible values for each hyperparameter, and then using a tuner (like RandomSearch or Hyperband) to search through this space.[51] It is important to start by defining a reasonable search space for the key hyperparameters, such as the learning rate, batch size, the choice of optimizer, and potentially architectural parameters like the number of filters in convolutional layers or the number of units in dense layers.[51] During the tuning process, it is essential to use a validation set to evaluate the performance of different hyperparameter combinations.[41] This helps in selecting hyperparameters that generalize well to unseen data and avoids overfitting to the training set. Given its efficiency, especially in terms of resource allocation and speed, considering the use of the Hyperband tuner within Keras Tuner is advisable.[53] It is important to be aware that hyperparameter tuning can

be a computationally intensive process, so it is necessary to plan accordingly based on the available computational resources.[54]

Properly Saving and Loading Trained TensorFlow/Keras Models

Once a satisfactory image classification model has been trained, it is essential to save it properly so that it can be loaded and used later for making predictions in the robotics application without the need for retraining.[71]

The recommended best practice for saving a Keras model is to save the entire model, which includes its architecture, the learned weights of the layers, the state of the optimizer, and the training configuration.[71] This comprehensive saving approach allows for the complete reconstruction of the model when it is loaded later.[73] The primary function used for this purpose is model.save(filepath), where filepath specifies the location where the model will be saved.[71] Saving the entire model in this way means that when it is loaded, it is ready to be used for prediction without requiring any additional steps like rebuilding the architecture or recompiling the model.[73] Keras offers flexibility in how models can be saved, supporting different file formats including the native .keras format (which is the recommended default in newer versions of TensorFlow), TensorFlow's SavedModel format (by setting save_format='tf'), and the legacy HDF5 format (by setting save_format='h5').[71]

To load a model that has been saved using model.save(), the tf.keras.models.load_model(filepath) function is used, where filepath is the path to the saved model file.[71] This function reconstructs the model exactly as it was when it was saved, including its architecture, weights, and optimizer state.[73] This makes it very convenient for deploying the trained model, as only a single file needs to be loaded. It is important to note that if the saved model includes any custom layers or custom functions, these might need to be explicitly provided to the load_model() function through the custom_objects argument, or they should be registered using @tf.keras.utils.register_keras_serializable() to ensure proper deserialization.[74] This is because TensorFlow needs to know how to recreate these custom components when loading the model.[74]

When considering saving and loading models for deployment in a robotics application, saving the entire model is generally the most straightforward approach as it encapsulates all the necessary information into a single file, simplifying the deployment process.[71] It is also important to ensure that the version of TensorFlow used for loading the model is compatible with the version that was used to save it, as inconsistencies between versions can sometimes lead to issues during the loading

process [General knowledge]. While there is also an option to save and load only the model's weights using model.save_weights() and model.load_weights(), this approach requires that the model architecture is defined and compiled separately before the weights can be loaded.[71] For most deployment scenarios, saving and loading the entire model using model.save() and tf.keras.models.load_model() is the more convenient and recommended method.

Best Practices for Preprocessing Input Images for Prediction

To ensure that the trained image classification model performs accurately when used in the robotics application, it is crucial to preprocess any new input images in exactly the same way that the images in the training dataset were preprocessed.[121] Any inconsistencies in the preprocessing steps can lead to a significant drop in the model's prediction accuracy.

There are several essential preprocessing steps that should be applied to any input image before it is fed into the trained model for prediction. One of the first steps is **resizing** the input image to the exact dimensions that the model was trained on.[73] For the CIFAR-10 dataset, the images are typically 32x32 pixels in size, so any new image needs to be resized to these dimensions.[73] This can be achieved using image processing libraries like OpenCV (with the cv2.resize() function) or using the image processing functions provided by TensorFlow (in the tf.image module, such as tf.image.resize()) [User Query]. Another critical preprocessing step is **normalization** of the pixel values.[109] The input image's pixel values must be normalized using the same method that was applied to the training data. For CIFAR-10, this usually involves scaling the pixel values to the range of 0 to 1 by dividing each pixel value by 255.0.[109] If the training data was in a specific **color space**, such as RGB, it is also important to ensure that the input image is in the same color space [User Query]. If an input image is in a different format (e.g., BGR, which is common with OpenCV), it should be converted to the correct color space using functions like cv2.cvtColor() in OpenCV.

Maintaining consistency in the preprocessing steps between training and prediction is of paramount importance.[121] The model learns to recognize patterns and features based on the way the training data was presented to it. If new input images are preprocessed differently, the model might not be able to correctly interpret them, leading to inaccurate predictions. To ensure this consistency, it is a good practice to encapsulate all the necessary preprocessing steps within the prediction function of the image classification model class. This helps in keeping the preprocessing pipeline consistent and reduces the chances of errors due to manual preprocessing

inconsistencies.

Here is an example of how a function to preprocess a single image might be implemented using OpenCV:

```python
import cv2 as cv
import numpy as np

def preprocess_image(image_path, target_size=(32, 32)):
    image = cv.imread(image_path)
    image = cv.resize(image, target_size)
    image = cv.cvtColor(image, cv.COLOR_BGR2RGB) # Assuming training was done with RGB
    image = image.astype('float32') / 255.0
    image = np.expand_dims(image, axis=0) # Add batch dimension
    return image
```

In this example, the function takes the path to an image, resizes it to the target size (defaulting to 32x32), converts its color space to RGB, normalizes the pixel values to the range , and then adds a batch dimension to the image array using np.expand_dims(). This is often necessary because Keras models, even when making a single prediction, expect the input to have a batch dimension.[121]

Conclusion and Recommendations for Robotics Application

In conclusion, to enhance the image classification capabilities for your robotics project, several key improvements can be implemented. Structuring the code as a modular Python class will lead to better organization and reusability. Adopting a more advanced CNN architecture, such as ResNet or DenseNet, holds the potential for achieving higher accuracy on the CIFAR-10 dataset. Integrating data augmentation using TensorFlow's ImageDataGenerator with appropriate transformations for CIFAR-10 will significantly improve the model's ability to generalize to unseen data and enhance its robustness. Applying proper normalization techniques, including pixel-wise normalization and potentially Batch Normalization, will ensure stable training and improved performance. Utilizing hyperparameter tuning strategies, especially with the help of Keras Tuner, will enable further optimization of the model's accuracy by finding the best settings for key hyperparameters. Following best

practices for saving and loading the trained model using model.save() and tf.keras.models.load_model() will facilitate easy and reliable deployment within the robotics application. Finally, adhering to consistent and correct preprocessing steps for any new input images before making predictions is crucial for ensuring accurate results.

To move forward with these enhancements, the following actionable steps are recommended:

1. Implement the suggested Python class structure to encapsulate the image classification functionality in your codebase.
2. Experiment with different CNN architectures, particularly ResNet or DenseNet variants, based on the guidance provided in this report, considering the trade-offs between accuracy and computational complexity for your robotics platform.
3. Integrate data augmentation into your training pipeline using ImageDataGenerator, applying relevant transformations such as random rotations, flips, shifts, and zooming, tailored for the CIFAR-10 dataset.
4. Ensure that pixel-wise normalization (dividing by 255.0) is applied to both your training and testing data. Consider adding Batch Normalization layers within your chosen CNN architecture to further improve training stability and model performance.
5. Explore hyperparameter tuning using the Keras Tuner library to find the optimal values for key hyperparameters like learning rate, batch size, optimizer, and potentially architectural parameters. Utilize a validation set to guide the tuning process.
6. Use the model.save() method to save your trained model in the recommended .keras format and tf.keras.models.load_model() to load it for use in your robotics application.
7. Implement the recommended preprocessing steps for any new input images before feeding them to the loaded model for prediction. This should include resizing to the correct input dimensions (32x32 for CIFAR-10), normalizing the pixel values, and converting the color space if necessary.
8. After implementing these improvements, thoroughly evaluate the performance of your enhanced model on a held-out test set to quantify the gains in accuracy.
9. Finally, integrate the trained and tested image classification model into the main control system of your robotics project, ensuring that the prediction functionality is robust and efficient, with appropriate error handling mechanisms in place.

By diligently following these recommendations, you should be well-positioned to

significantly improve the structure, accuracy, and overall reliability of your image classification model for your robotics application, enabling more effective perception and interaction with the environment.

Works cited

1. Keras and TensorFlow Mastery: Best Practices and Tips - GPTutorPro, accessed on April 30, 2025, https://gpttutorpro.com/keras-and-tensorflow-mastery-best-practices-and-tips/
2. TensorFlow 2.0 + Keras Crash course - Kaggle, accessed on April 30, 2025, https://www.kaggle.com/code/kurianbenoy/tensorflow-2-0-keras-crash-course
3. Does anyone have some tf/Keras best practices lists or examples? : r/tensorflow - Reddit, accessed on April 30, 2025, https://www.reddit.com/r/tensorflow/comments/kdc8ft/does_anyone_have_some_tfkeras_best_practices/
4. Introduction to modules, layers, and models | TensorFlow Core, accessed on April 30, 2025, https://www.tensorflow.org/guide/intro_to_modules
5. 3 ways to create a Keras model with TensorFlow 2.0 (Sequential, Functional, and Model Subclassing) - PyImageSearch, accessed on April 30, 2025, https://pyimagesearch.com/2019/10/28/3-ways-to-create-a-keras-model-with-tensorflow-2-0-sequential-functional-and-model-subclassing/
6. Best practices to write Deep Learning code: Project structure, OOP, Type checking and documentation | AI Summer, accessed on April 30, 2025, https://theaisummer.com/best-practices-deep-learning-code/
7. Convolutional Neural Networks for CIFAR-10 | cifar-10-cnn - Wei Li, accessed on April 30, 2025, https://bigballon.github.io/cifar-10-cnn/
8. Cifar-10 Images Classification using CNNs (88%) - Kaggle, accessed on April 30, 2025, https://www.kaggle.com/code/faressayah/cifar-10-images-classification-using-cnns-88
9. yogeshgajjar/CNN-CIFAR-10-image-classification: CNN based CIFAR-10 Image Classifier using All-CNN (YGNet) architecture (90% accuracy) and LeNet-5 architecture (74% accuracy) - GitHub, accessed on April 30, 2025, https://github.com/yogeshgajjar/CNN-CIFAR-10-image-classification
10. Best identified CNN architectures on CIFAR-10 - ResearchGate, accessed on April 30, 2025, https://www.researchgate.net/figure/Best-identified-CNN-architectures-on-CIFAR-10_tbl3_332412530
11. Well-tuned CNN architecture for CIFAR-10 dataset. Left: This network is... - ResearchGate, accessed on April 30, 2025, https://www.researchgate.net/figure/Well-tuned-CNN-architecture-for-CIFAR-10-dataset-Left-This-network-is-generated-by_fig3_326816043
12. neural network - Conv-2 CNN architecture - CIFAR-10 - Data Science Stack Exchange, accessed on April 30, 2025, https://datascience.stackexchange.com/questions/68235/conv-2-cnn-architectur

e-cifar-10
13. Conv-2 CNN architecture - CIFAR-10 : r/MLQuestions - Reddit, accessed on April 30, 2025, https://www.reddit.com/r/MLQuestions/comments/f59a9k/conv2_cnn_architecture_cifar10/
14. deep-learning-v2-pytorch/convolutional-neural-networks/cifar-cnn/cifar10_cnn_exercise.ipynb at master - GitHub, accessed on April 30, 2025, https://github.com/WillKoehrsen/deep-learning-v2-pytorch/blob/master/convolutional-neural-networks/cifar-cnn/cifar10_cnn_exercise.ipynb
15. [D] Where can I find CNN baseline architectures for MNIST, CIFAR-10? - Reddit, accessed on April 30, 2025, https://www.reddit.com/r/MachineLearning/comments/cuaqdq/d_where_can_i_find_cnn_baseline_architectures_for/
16. Data Augmentation for Improving Image Classification Accuracy - Data Annotation Platform, accessed on April 30, 2025, https://keylabs.ai/blog/data-augmentation-for-improving-image-classification-accuracy/
17. Guide To Data Augmentation For Image Classification - Averroes AI, accessed on April 30, 2025, https://averroes.ai/blog/guide-to-data-augmentation-for-image-classification
18. Data Augmentation in Classification and Segmentation: A Survey and New Strategies - PMC, accessed on April 30, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC9966095/
19. A Complete Guide to Data Augmentation | DataCamp, accessed on April 30, 2025, https://www.datacamp.com/tutorial/complete-guide-data-augmentation
20. Data Augmentation for Image Classification using Generative AI - arXiv, accessed on April 30, 2025, https://arxiv.org/html/2409.00547v1
21. Why use Data Augmentation? : r/MLQuestions - Reddit, accessed on April 30, 2025, https://www.reddit.com/r/MLQuestions/comments/1ay7ci5/why_use_data_augmentation/
22. How data augmentation can improve ML model accuracy | JFrog ML - Qwak, accessed on April 30, 2025, https://www.qwak.com/post/how-data-augmentation-can-improve-ml-model-accuracy
23. The Effectiveness of Data Augmentation in Image Classification using Deep Learning - CS231n, accessed on April 30, 2025, https://cs231n.stanford.edu/reports/2017/pdfs/300.pdf
24. Data Augmentation Helps Improve Model Accuracy - Valohai, accessed on April 30, 2025, https://valohai.com/blog/data-augmentation/
25. Data Augmentation Cifar 10 Techniques | Restackio, accessed on April 30, 2025, https://www.restack.io/p/data-augmentation-answer-cifar-10-cat-ai
26. DATA AUGMENTATION TECHNIQUES USING CIFAR10 DATASET and FMNIST - NORMA@NCI Library, accessed on April 30, 2025, https://norma.ncirl.ie/7163/2/prachiconfigurationmanual.pdf

27. Data Augmentation Cifar10 Pytorch | Restackio, accessed on April 30, 2025, https://www.restack.io/p/data-augmentation-knowledge-cifar10-answer-cat-ai
28. CIFAR-10 Benchmark (Data Augmentation) | Papers With Code, accessed on April 30, 2025, https://paperswithcode.com/sota/data-augmentation-on-cifar-10
29. CIFAR-10 Image Classification with CNN - Kaggle, accessed on April 30, 2025, https://www.kaggle.com/code/farzadnekouei/cifar-10-image-classification-with-cnn
30. Modern Data Augmentation Techniques for Computer Vision | tfaugmentation - Wandb, accessed on April 30, 2025, https://wandb.ai/authors/tfaugmentation/reports/Modern-Data-Augmentation-Techniques-for-Computer-Vision--VmlldzoxNzU3NTU
31. moritzhambach/Image-Augmentation-in-Keras-CIFAR-10 - GitHub, accessed on April 30, 2025, https://github.com/moritzhambach/Image-Augmentation-in-Keras-CIFAR-10-
32. Machine-Learning-Notebooks/Recognizing-CIFAR-10-images-Improved-Model-Data-Augmentation.ipynb at master - GitHub, accessed on April 30, 2025, https://github.com/chhayac/Machine-Learning-Notebooks/blob/master/Recognizing-CIFAR-10-images-Improved-Model-Data-Augmentation.ipynb
33. Numerical data: Normalization | Machine Learning - Google for Developers, accessed on April 30, 2025, https://developers.google.com/machine-learning/crash-course/numerical-data/normalization
34. What is Normalization in Machine Learning? A Comprehensive Guide to Data Rescaling, accessed on April 30, 2025, https://www.datacamp.com/tutorial/normalization-in-machine-learning
35. Understanding the Impact of Batch Normalization on CNNs - TiDB, accessed on April 30, 2025, https://www.pingcap.com/article/understanding-the-impact-of-batch-normalization-on-cnns/
36. A Two-Step Data Normalization Approach for Improving Classification Accuracy in the Medical Diagnosis Domain - MDPI, accessed on April 30, 2025, https://www.mdpi.com/2227-7390/10/11/1942
37. Brain-inspired Weighted Normalization for CNN Image Classification - bioRxiv, accessed on April 30, 2025, https://www.biorxiv.org/content/10.1101/2021.05.20.445029v1.full-text
38. Is normalization indispensable for training deep neural networks?, accessed on April 30, 2025, https://proceedings.neurips.cc/paper/2020/file/9b8619251a19057cff70779273e95aa6-Paper.pdf
39. Normalization vs standardization for image classification problem, accessed on April 30, 2025, https://datascience.stackexchange.com/questions/94537/normalization-vs-standardization-for-image-classification-problem
40. Highly Efficient and Accurate Deep Learning–Based Classification of MRI Contrast on a CPU and GPU - PMC, accessed on April 30, 2025,

https://pmc.ncbi.nlm.nih.gov/articles/PMC9156587/
41. Hyperparameter Tuning for Image Classification | Restackio, accessed on April 30, 2025, https://www.restack.io/p/hyperparameter-tuning-answer-image-classification-cat-ai
42. Hyperparameter tuning | GeeksforGeeks, accessed on April 30, 2025, https://www.geeksforgeeks.org/hyperparameter-tuning/
43. Mastering Hyperparameter Tuning Techniques to Boost ML Models, accessed on April 30, 2025, https://www.numberanalytics.com/blog/mastering-hyperparameter-tuning-techniques-ml-models
44. Tuning the Hyperparameters and Layers of Neural Network Deep Learning, accessed on April 30, 2025, https://www.analyticsvidhya.com/blog/2021/05/tuning-the-hyperparameters-and-layers-of-neural-network-deep-learning/
45. Image Classification Hyperparameters - Amazon SageMaker AI, accessed on April 30, 2025, https://docs.aws.amazon.com/sagemaker/latest/dg/IC-Hyperparameter.html
46. Fine-tuning Models: Hyperparameter Optimization - Encord, accessed on April 30, 2025, https://encord.com/blog/fine-tuning-models-hyperparameter-optimization/
47. Understand the hyperparameter tuning strategies available in Amazon SageMaker AI, accessed on April 30, 2025, https://docs.aws.amazon.com/sagemaker/latest/dg/automatic-model-tuning-how-it-works.html
48. What's the right way of doing hyperparameter tuning when solving problems using deep RL? - Reddit, accessed on April 30, 2025, https://www.reddit.com/r/reinforcementlearning/comments/hbdcs1/whats_the_right_way_of_doing_hyperparameter/
49. What's your strategy for hyperparameter tuning : r/computervision - Reddit, accessed on April 30, 2025, https://www.reddit.com/r/computervision/comments/1fi2iub/whats_your_strategy_for_hyperparameter_tuning/
50. Any tips on hyperparameter tuning? - Deep Learning - fast.ai Course Forums, accessed on April 30, 2025, https://forums.fast.ai/t/any-tips-on-hyperparameter-tuning/73090
51. Automatic Hyperparameter Optimization With Keras Tuner - DigitalOcean, accessed on April 30, 2025, https://www.digitalocean.com/community/tutorials/hyperparameter-optimization-with-keras-tuner
52. Learning Rate Scheduler in Keras and TensorFlow - - YouTube, accessed on April 30, 2025, https://www.youtube.com/watch?v=mKt_m0er8jg
53. Introduction to the Keras Tuner | TensorFlow Core, accessed on April 30, 2025, https://www.tensorflow.org/tutorials/keras/keras_tuner
54. Easy Hyperparameter Tuning with Keras Tuner and TensorFlow - PyImageSearch,

accessed on April 30, 2025, https://pyimagesearch.com/2021/06/07/easy-hyperparameter-tuning-with-keras-tuner-and-tensorflow/
55. Hyperparameter tuning for Deep Learning with scikit-learn, Keras, and TensorFlow, accessed on April 30, 2025, https://pyimagesearch.com/2021/05/31/hyperparameter-tuning-for-deep-learning-with-scikit-learn-keras-and-tensorflow/
56. Automated hyper-parameter tuning | TensorFlow Decision Forests, accessed on April 30, 2025, https://www.tensorflow.org/decision_forests/tutorials/automatic_tuning_colab
57. How to tune learning rate with HParams Dashboard on Tensorflow?, accessed on April 30, 2025, https://datascience.stackexchange.com/questions/75102/how-to-tune-learning-rate-with-hparams-dashboard-on-tensorflow
58. Easy Hyperparameter Tuning in Neural Networks using Keras Tuner - Analytics Vidhya, accessed on April 30, 2025, https://www.analyticsvidhya.com/blog/2021/08/easy-hyperparameter-tuning-in-neural-networks-using-keras-tuner/
59. How to choose optimizer and learning rate for hyperparameter training in keras-tuner, accessed on April 30, 2025, https://stackoverflow.com/questions/62684535/how-to-choose-optimizer-and-learning-rate-for-hyperparameter-training-in-keras-t
60. How to Grid Search Hyperparameters for Deep Learning Models in Python with Keras, accessed on April 30, 2025, https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/
61. Tune hyperparameters in your custom training loop - Keras, accessed on April 30, 2025, https://keras.io/keras_tuner/guides/custom_tuner/
62. Hyperparameter tuning for dropout, # neurons, batch size, # epochs, and weight constraint, accessed on April 30, 2025, https://m.youtube.com/watch?v=Bzsxq1JJbbo&pp=ygUIl2tlcmFzY3Y%3D
63. Hyperparameter Tuning with Keras / Tensorflow for multivariate time series regression, accessed on April 30, 2025, https://stats.stackexchange.com/questions/498276/hyperparameter-tuning-with-keras-tensorflow-for-multivariate-time-series-regre
64. How to tune the number of epochs and batch_size? · Issue #122 · keras-team/keras-tuner, accessed on April 30, 2025, https://github.com/keras-team/keras-tuner/issues/122
65. [D] What is the tradeoff or relation between batch size and the number of epochs? - Reddit, accessed on April 30, 2025, https://www.reddit.com/r/MachineLearning/comments/igfej1/d_what_is_the_tradeoff_or_relation_between_batch/
66. Where do # of epochs and batch size belong in the hyperparameter tuning process?, accessed on April 30, 2025, https://stackoverflow.com/questions/61490006/where-do-of-epochs-and-batch-

size-belong-in-the-hyperparameter-tuning-process
67. Integrate batch size in keras tuner - python - Stack Overflow, accessed on April 30, 2025, https://stackoverflow.com/questions/76109047/integrate-batch-size-in-keras-tuner
68. Image classification | TensorFlow Core, accessed on April 30, 2025, https://www.tensorflow.org/tutorials/images/classification
69. machine-learning-articles/how-to-predict-new-samples-with-your-keras-model.md at main, accessed on April 30, 2025, https://github.com/christianversloot/machine-learning-articles/blob/main/how-to-predict-new-samples-with-your-keras-model.md
70. Python | Image Classification using Keras - GeeksforGeeks, accessed on April 30, 2025, https://www.geeksforgeeks.org/python-image-classification-using-keras/
71. Save and load models | TensorFlow Core, accessed on April 30, 2025, https://www.tensorflow.org/tutorials/keras/save_and_load
72. How to Save and Load Your Keras Deep Learning Model - MachineLearningMastery.com, accessed on April 30, 2025, https://machinelearningmastery.com/save-load-keras-deep-learning-models/
73. How to use a saved model in Keras to predict and classify an image? - Stack Overflow, accessed on April 30, 2025, https://stackoverflow.com/questions/50151585/how-to-use-a-saved-model-in-keras-to-predict-and-classify-an-image
74. Save / Load Tensorflow Keras model for Prediction only - Stack Overflow, accessed on April 30, 2025, https://stackoverflow.com/questions/61894658/save-load-tensorflow-keras-model-for-prediction-only
75. Image Classification with Keras - Nextjournal, accessed on April 30, 2025, https://nextjournal.com/mpd/image-classification-with-keras
76. Save and load models in Tensorflow - GeeksforGeeks, accessed on April 30, 2025, https://www.geeksforgeeks.org/save-and-load-models-in-tensorflow/
77. Whole model saving & loading - Keras, accessed on April 30, 2025, https://keras.io/2/api/models/model_saving_apis/model_saving_and_loading/
78. Save and load a model using a distribution strategy | TensorFlow Core, accessed on April 30, 2025, https://www.tensorflow.org/tutorials/distribute/save_and_load
79. How to properly save and load model weights - Keras - TensorFlow Forum, accessed on April 30, 2025, https://discuss.ai.google.dev/t/how-to-properly-save-and-load-model-weights/23916
80. Save, serialize, and export models | TensorFlow Core, accessed on April 30, 2025, https://www.tensorflow.org/guide/keras/serialization_and_saving
81. save and load keras model with custom layer with additional attributes - Stack Overflow, accessed on April 30, 2025, https://stackoverflow.com/questions/63083440/save-and-load-keras-model-with-custom-layer-with-additional-attributes
82. How to load a tensorflow keras model saved with saved_model to use the predict

function?, accessed on April 30, 2025, https://stackoverflow.com/questions/70520424/how-to-load-a-tensorflow-keras-model-saved-with-saved-model-to-use-the-predict-f
83. How to use a model to do predictions with Keras - ActiveState, accessed on April 30, 2025, https://www.activestate.com/resources/quick-reads/how-to-use-a-model-to-do-predictions-with-keras/
84. How to predict after creating keras model (done after 'complie', 'fit', 'history'..)?, accessed on April 30, 2025, https://forum.posit.co/t/how-to-predict-after-creating-keras-model-done-after-complie-fit-history/27290
85. How to Make Predictions with Keras - MachineLearningMastery.com, accessed on April 30, 2025, https://machinelearningmastery.com/how-to-make-classification-and-regression-predictions-for-deep-learning-models-in-keras/
86. Making predictions - Decision Forests - TensorFlow, accessed on April 30, 2025, https://www.tensorflow.org/decision_forests/tutorials/predict_colab
87. Passing parameters to model.predict in tf.keras.Model - Stack Overflow, accessed on April 30, 2025, https://stackoverflow.com/questions/66750258/passing-parameters-to-model-predict-in-tf-keras-model
88. Using model.predict and interpreting results - Keras - Google AI Developers Forum, accessed on April 30, 2025, https://discuss.ai.google.dev/t/using-model-predict-and-interpreting-results/26151
89. Confusion about keras Model: __call__ vs. call vs. predict methods - Stack Overflow, accessed on April 30, 2025, https://stackoverflow.com/questions/60837962/confusion-about-keras-model-call-vs-call-vs-predict-methods
90. tf.keras.Model | TensorFlow v2.16.1, accessed on April 30, 2025, https://www.tensorflow.org/api_docs/python/tf/keras/Model
91. Your First Deep Learning Project in Python with Keras Step-by-Step - MachineLearningMastery.com, accessed on April 30, 2025, https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/
92. python - model.predict_classes is deprecated - What to use instead? - Stack Overflow, accessed on April 30, 2025, https://stackoverflow.com/questions/68776790/model-predict-classes-is-deprecated-what-to-use-instead
93. Why the resnet110 I train on CIFAR10 dataset only get 77% test acc - Stack Overflow, accessed on April 30, 2025, https://stackoverflow.com/questions/58986583/why-the-resnet110-i-train-on-cifar10-dataset-only-get-77-test-acc
94. Cifar-10 classification with ResNet-20: training loss (a), test... - ResearchGate, accessed on April 30, 2025, https://www.researchgate.net/figure/Cifar-10-classification-with-ResNet-20-train

ing-loss-a-test-accuracy-b-and-rescaled_fig1_351781715
95. 94% on CIFAR-10 in 94 lines and 94 seconds | The Good Minima, accessed on April 30, 2025, https://johanwind.github.io/2022/12/28/cifar_94.html
96. Need help for improving accuracy on cifar 10 datasets - Part 1 (2020) - Fast.ai Forums, accessed on April 30, 2025, https://forums.fast.ai/t/need-help-for-improving-accuracy-on-cifar-10-datasets/85105
97. CIFAR-10 Benchmark (Image Classification) - Papers With Code, accessed on April 30, 2025, https://paperswithcode.com/sota/image-classification-on-cifar-10
98. VGG Transfer Learning with CIFAR-10 - Kaggle, accessed on April 30, 2025, https://www.kaggle.com/code/hossamahmedsalah/vgg-transfer-learning-with-cifar-10
99. Accuracy of a VGG-19 network trained in CIFAR-10 with different regularisation techniques., accessed on April 30, 2025, https://www.researchgate.net/figure/Accuracy-of-a-VGG-19-network-trained-in-CIFAR-10-with-different-regularisation-techniques_tbl1_324584468
100. Low validation accuracy VGG-19 CIFAR-10 CNN : r/deeplearning - Reddit, accessed on April 30, 2025, https://www.reddit.com/r/deeplearning/comments/hxk3g8/low_validation_accuracy_vgg19_cifar10_cnn/
101. Got always about 10% accuracy when training CIFAR-10 by VGG-16 model #1968 - GitHub, accessed on April 30, 2025, https://github.com/apache/incubator-mxnet/issues/1968
102. Training VGG16 with CIFAR-10 dataset in TensorFlow, can't get good accuracy, accessed on April 30, 2025, https://stackoverflow.com/questions/53476098/training-vgg16-with-cifar-10-dataset-in-tensorflow-cant-get-good-accuracy
103. Why is VGG-16 performing poor on CIFAR-10 dataset? - Stack Overflow, accessed on April 30, 2025, https://stackoverflow.com/questions/68323230/why-is-vgg-16-performing-poor-on-cifar-10-dataset
104. CIFAR-10 on Benchmarks.AI, accessed on April 30, 2025, https://benchmarks.ai/cifar-10
105. Why does adding data augmentation decrease training accuracy a tiny bit?, accessed on April 30, 2025, https://datascience.stackexchange.com/questions/95019/why-does-adding-data-augmentation-decrease-training-accuracy-a-tiny-bit
106. python 3.x - CIFAR10 example : Keras - Stack Overflow, accessed on April 30, 2025, https://stackoverflow.com/questions/54606300/cifar10-example-keras
107. kolbydboyd/CIFAR-10-Image-Classification-using-TensorFlow-and-Keras - GitHub, accessed on April 30, 2025, https://github.com/kolbydboyd/CIFAR-10-Image-Classification-using-TensorFlow-and-Keras
108. Keras Cifar10 Example — Ray 2.44.1, accessed on April 30, 2025, https://docs.ray.io/en/latest/tune/examples/includes/pbt_tune_cifar10_with_keras.

html
109. CIFAR-10 Image Classification in TensorFlow - GeeksforGeeks, accessed on April 30, 2025, https://www.geeksforgeeks.org/cifar-10-image-classification-in-tensorflow/
110. CIFAR 10 88% Accuracy using Keras - Kaggle, accessed on April 30, 2025, https://www.kaggle.com/code/kedarsai/cifar-10-88-accuracy-using-keras
111. CIFAR-10 Keras image data augmentation effect for one image only - Stack Overflow, accessed on April 30, 2025, https://stackoverflow.com/questions/56710332/cifar-10-keras-image-data-augmentation-effect-for-one-image-only
112. CNN with Image Augmentation (CIFAR10).ipynb - GitHub, accessed on April 30, 2025, https://github.com/moritzhambach/Image-Augmentation-in-Keras-CIFAR-10-/blob/master/CNN%20with%20Image%20Augmentation%20(CIFAR10).ipynb
113. Why does normalizing MNIST images reduce accuracy? - Stack Overflow, accessed on April 30, 2025, https://stackoverflow.com/questions/60621302/why-does-normalizing-mnist-images-reduce-accuracy
114. Basic classification: Classify images of clothing | TensorFlow Core, accessed on April 30, 2025, https://www.tensorflow.org/tutorials/keras/classification
115. Keras Tuner - Kaggle, accessed on April 30, 2025, https://www.kaggle.com/code/mathurutkarsh/keras-tuner
116. CIFAR10 Using Keras Tuner 97% accuracy - Kaggle, accessed on April 30, 2025, https://www.kaggle.com/code/shrijoychowdhury/cifar10-using-keras-tuner-97-accuracy
117. Hyperparameter tuning with Keras Tuner - The TensorFlow Blog, accessed on April 30, 2025, https://blog.tensorflow.org/2020/01/hyperparameter-tuning-with-keras-tuner.html
118. clearml/examples/frameworks/kerastuner/keras_tuner_cifar.py at master - GitHub, accessed on April 30, 2025, https://github.com/allegroai/clearml/blob/master/examples/frameworks/kerastuner/keras_tuner_cifar.py
119. CIFAR-10 Classifiers: Part 2 - Use Keras Tuner to speed up the hyperparameter search - Kode Bytes, accessed on April 30, 2025, https://kb.katnoria.com/posts/2020/8/tf2-keras-tuner/
120. Tensorflow - Save & Load custom model in Keras v3 format, accessed on April 30, 2025, https://discuss.ai.google.dev/t/tensorflow-save-load-custom-model-in-keras-v3-format/32372
121. Image Classification using Pre-Trained ImageNet Models in TensorFlow & Keras, accessed on April 30, 2025, https://learnopencv.com/image-classification-pretrained-imagenet-models-tensorflow-keras/