To make a Convolutional Neural Network (CNN) that classifies 10 kinds of animals more explainable through the usage of feature visualization, you can follow these steps:

1. Train the CNN: First, you need to train your CNN using a labeled dataset of animal images. This dataset should contain images of the 10 different kinds of animals you want to classify. Train your CNN using standard techniques such as backpropagation and gradient descent to optimize its weights and biases.

2. Choose a Layer for Feature Visualization: Select a layer in your CNN that you want to visualize. Generally, the earlier layers in the network capture low-level features such as edges and textures, while the later layers capture higher-level features like shapes and object parts. Depending on the level of interpretability you desire, you can choose a layer accordingly.

3. Define an Optimization Objective: To generate interpretable visualizations, you need to define an optimization objective that encourages the network to generate meaningful images. One common approach is to maximize the activation of a particular neuron in the chosen layer. By optimizing the input image to maximize the activation of a specific neuron, you can visualize what kind of patterns or features that neuron is sensitive to.

4. Perform Gradient Ascent: Utilize gradient ascent to update the input image iteratively. Start with a random or noise-filled image, and then compute the gradient of the chosen neuron's activation with respect to the input image pixels. Adjust the input image pixels in the direction that increases the activation, while ensuring that the image remains visually interpretable.

5. Regularization and Constraints: To ensure that the generated image remains visually meaningful, you can apply regularization techniques. For instance, you can introduce regularization terms that penalize high-frequency components or limit the total variation of the image. Additionally, you can impose constraints such as pixel value range or spatial smoothness to avoid unrealistic images.

6. Iterate and Visualize: Repeat the gradient ascent process for multiple iterations, adjusting the input image based on the gradient, until you achieve a visually interpretable representation. Monitor the progress and visualize the intermediate images during the iterations to understand how the features are being captured and refined.

7. Interpretation: Once you have obtained the feature visualizations, analyze the generated images to gain insights into what the network has learned. Look for consistent patterns, shapes, or textures that are present across different class-specific visualizations. These features can provide an understanding of what the network is focusing on when distinguishing between different animal classes.

By utilizing feature visualization techniques, you can gain insights into the learned features of your CNN, making it more explainable and interpretable. This can help in understanding the decision-making process of the network and validating its classifications.

Article points:

Feature visualization is about getting insights into which features parts of a neural network are looking for by generating examples.

Feature visualization by optimization: starting from random noise we optimize an image to activate neuron(s) in a network.

Optimization isolates the causes of behavior from mere correlations. A neuron may not be detecting what you initially thought.

Lack of diversity: optimization usually gives an extremely positive example.

We find there’s a very simple way to achieve diversity: adding a “diversity term”

Examples like these suggest that neurons are not necessarily the right semantic units for understanding neural nets.

Strided convolutions and pooling operations, which create high-frequency patterns in the gradient

Dealing with this high frequency noise has been one of the primary challenges and overarching threads of feature visualization research. If you want to get useful visualizations, you need to impose a more natural structure using some kind of prior, regularizer, or constraint.

Regularization options:

**Frequency penalization**: targets the high-frequency patterns.

**Transformation robustness** tries to find examples that still activate the optimization target highly even if we slightly transform them by stochastically jittering, rotating, or scaling the image before the optimization step.

**Learned priors**. Our previous regularizers use very simple heuristics to keep examples reasonable. A natural next step is to actually learn a model of the real data and try to enforce that. With a strong model, this becomes similar to searching over the dataset. This approach produces the most photorealistic visualizations, but it may be unclear what came from the model being visualized and what came from the prior.

Preconditioning = transforming the gradient