Core Image

CI Filters

Step 0 (lol): Define a context in which Core Image can do the work of filtering images

`let context = CIContext(options: nil)`

Step 1: Configure a method that you can pass images through to filter/alter.

`func chromeEffectFilter(_ image: UIImage) -> UIImage { }`

Step 2: Define a filter that you want to use. You can find the listings here: Core Image Filter Reference

```
func chromeEffectFilter(_ image: UIImage) -> UIImage {
  let filter = CIFilter(name: "CIPhotoEffectChrome")
}
```

Step 3: Images need to be of type CIImage in order to be passed through any Core Image filter.

```
func chromeEffectFilter(_ image: UIImage) -> UIImage {
  let filter = CIFilter(name: "CIPhotoEffectChrome")
  let originalImage = CIImage(image: image)
}
```

Step 4: Now that the image is of the right type, we can apply the filter and any potential settings it might have like saturation, brightness, radius, etc. Use dot notation to access the `setValue` initializer to modify what you need. You'll always need to use the `inputImage` parameter at the very least.

```
func chromeEffectFilter(_ image: UIImage) -> UIImage {
  let filter = CIFilter(name: "CIPhotoEffectChrome")
  let originalImage = CIImage(image: image)
  filter.setValue(originalImage, forKey: kCIInputImageKey)
}
```

Step 5: Once the image is of type CIImage and the filter is set up the way you want, the actual work of processing the image through the filter can occur via Core Graphics in the Core Image context you defined earlier. It's sort of a two step process:

```
func chromeEffectFilter(_ image: UIImage) -> UIImage {
  let filter = CIFilter(name: "CIPhotoEffectChrome")
  let ciImage = CIImage(image: image)
  filter.setValue(originalImage, forKey: kCIInputImageKey)
  guard let outputImage = filter.outputImage,
      let cgImage = context.createCGImage(outputImage, from:
outputImage.extent) else { return nil }
}
```

Step 6: Assuming that all went well with your filter configuration (be sure your filter name String matches the documentation exactly!), your `cgImage` should be filtered according to what you specified. The function can return this filtered image that you'll be able to display in your app.

```
func chromeEffectFilter(_ image: UIImage) -> UIImage {
  let filter = CIFilter(name: "CIPhotoEffectChrome")
  let ciImage = CIImage(image: image)
  filter.setValue(originalImage, forKey: kCIInputImageKey)
  guard let outputImage = filter.outputImage,
      let cgImage = context.createCGImage(outputImage, from:
outputImage.extent) else { return nil }

  return UIImage(cgImage: cgImage)
}
```

Using the Image Picker

Now that you have some sort of method to send images through to filter them, you can use existing images or ones that the user captures with their camera.

Step 1: Drag out a UIButton to create an IBAction where the Image Picker will appear when the user taps it

```
@IBAction func selectImage(_ sender: Any) { }
```

Step 2: Be sure your class conforms to the `UIImagePickerControllerDelegate` protocol so that you can use its necessary methods. Once that's squared away, you can configure a pop up to let them pick between their camera or their Photos Library for picking their image once you've declared an instance of a `UIImagePickerController`. Be sure to set the delegate of the Image Picker Controller to the view that you're working in.

```
@IBAction func selectImage(_ sender: Any) {
  let imagePickerController = UIImagePickerController()
  let alert = UIAlertController(title: “How would you like to select a photo?”, message: “Pick a source:”, preferredStyle: .actionSheet)
  let imagePickerController.delegate = self
}
```

Step 3: `UIImagePickerController` has a property called `.sourceType` that will let you select from a number of enums to use. Let's have an alert present the Photos Library that a user can choose photos from.

```
@IBAction func selectImage(_ sender: Any) {
  let imagePickerController = UIImagePickerController()
  let alert = UIAlertController(title: “How would you like to select a photo?”, message: “Pick a source:”, preferredStyle: .actionSheet)
  let imagePickerController.delegate = self
  if UIImagePickerController.isSourceTypeAvailable(.photoLibrary) {
            let photoLibraryAction = UIAlertAction(title: "Photo Library", style: .default) { (_) in

                imagePickerController.sourceType = .photoLibrary
                self.present(imagePickerController, animated: true, completion: nil)
            }
            alert.addAction(photoLibraryAction)
        }
}
```

*note that it’s probably a good idea to implement some sort of cancel call on the alert just in case the user decides to not to pick a photo after all.

Step 4: Assuming that you have some sort of `UIImageView` to display the image, we can use a delegate method in from `UIImagePickerController` to do something with a photo that the user has selected. The `didFinishPickingMediaWithInfo` is going to be super useful here:

```
func imagePickerController(_ picker: UIImagePickerController,
didFinishPickingMediaWithInfo info: [UIImagePickerController.InfoKey : Any]) {

}
```

Step 5: Use the method's parameters to solidify the selection as a `UIImage` that we can use in our filter function.

```
func imagePickerController(_ picker: UIImagePickerController,
didFinishPickingMediaWithInfo info: [UIImagePickerController.InfoKey : Any]) {
  guard let image = info[.originalImage] as? UIImage else { return }
  let filteredImage = chromeEffectFilter(image)
}
```

Step 6: Now that the user has picked a photo and our custom filter function has altered it in some way, we can go ahead and display the image in our UI and dismiss the Image Picker Controller.

```
func imagePickerController(_ picker: UIImagePickerController,
didFinishPickingMediaWithInfo info: [UIImagePickerController.InfoKey : Any]) {
  guard let image = info[.originalImage] as? UIImage else { return }
  let filteredImage = chromeEffectFilter(image)
  imageView.image = filteredImage
  picker.dismiss(animate: true, completion: nil)
}
```

*it might be a good idea to disable or hide the `selectImageButton` at this point :)

And that's really all there is to filtering a photo via an Image Picker Controller from the Photos Library!