Animal Face Classification using CNN

By Keith S

DTSA -5511 Introduction To Deep Learning

Final Project

Introduction

- This is the final project for DTSA-5511 Introduction to Deep Learning. The purpose of this project is to develop a convolutional neural network to categorize animal pictures in to one of three categories; Cat, Dog, or Wild. This project is broken down into 3 main sections. First is initial exploratory data analysis of the dataset. Second is the design, build, training, and tuning of the CNN model. Third is a look into how the model is working. A closer look at the convolution filters and feature extraction being performed on the images.

DataSet

- Dataset used is Animal Faces-HQ (AFHQ) a collection of animal faces catagorized as Dog, Cat, or Wild. Dataset was created in support of the research paper 'StarGAN v2: Diverse Image Synthesis for Multiple Domains'. All pictures are 512x512 resolution in RGB color. There is a total of 15803 png images.

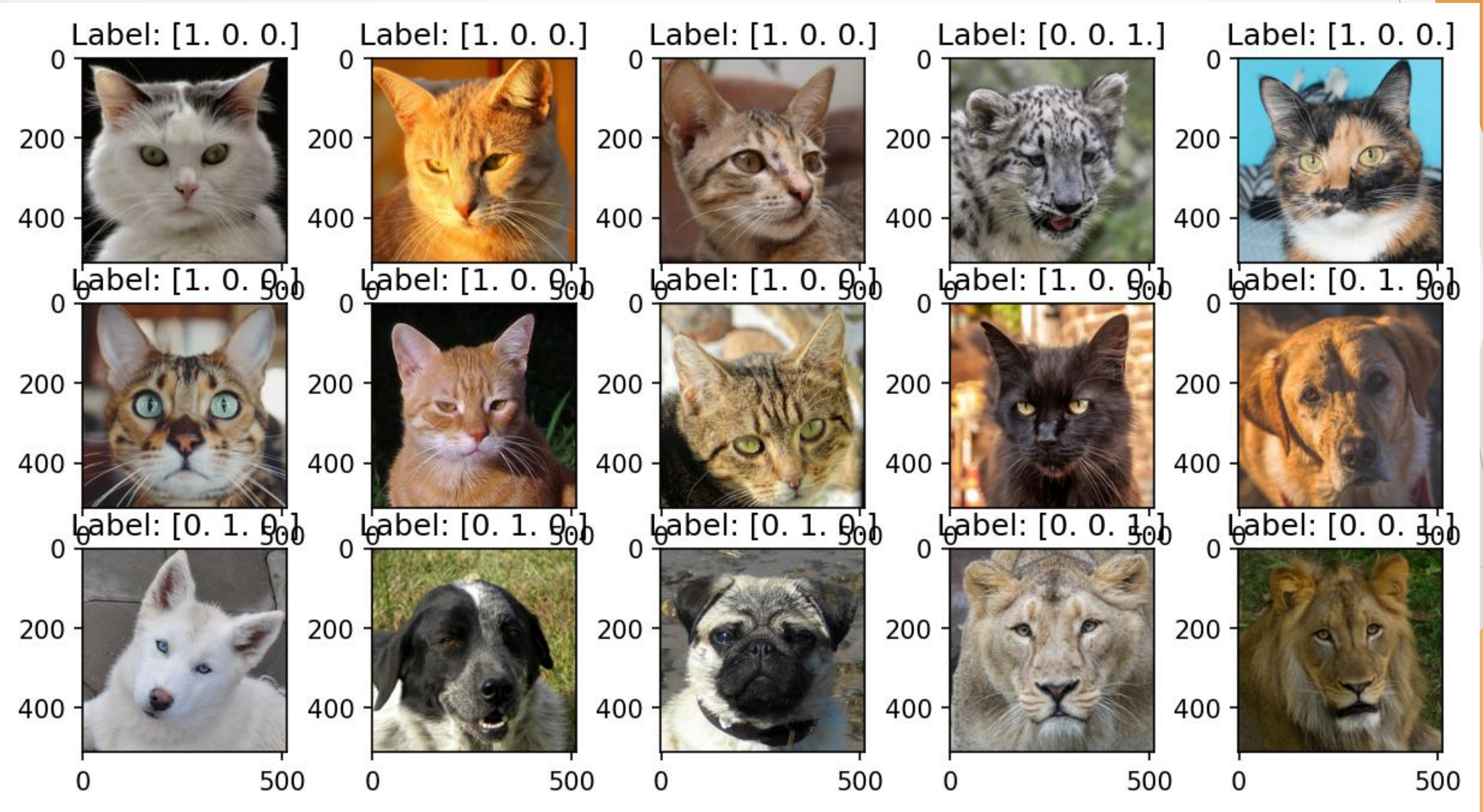
Dataset provided by:

- Choi, Y., Uh , Y., Yoo , J., & Ha, J.-W. (2020). StarGAN v2: Diverse Image Synthesis for Multiple Domains. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.

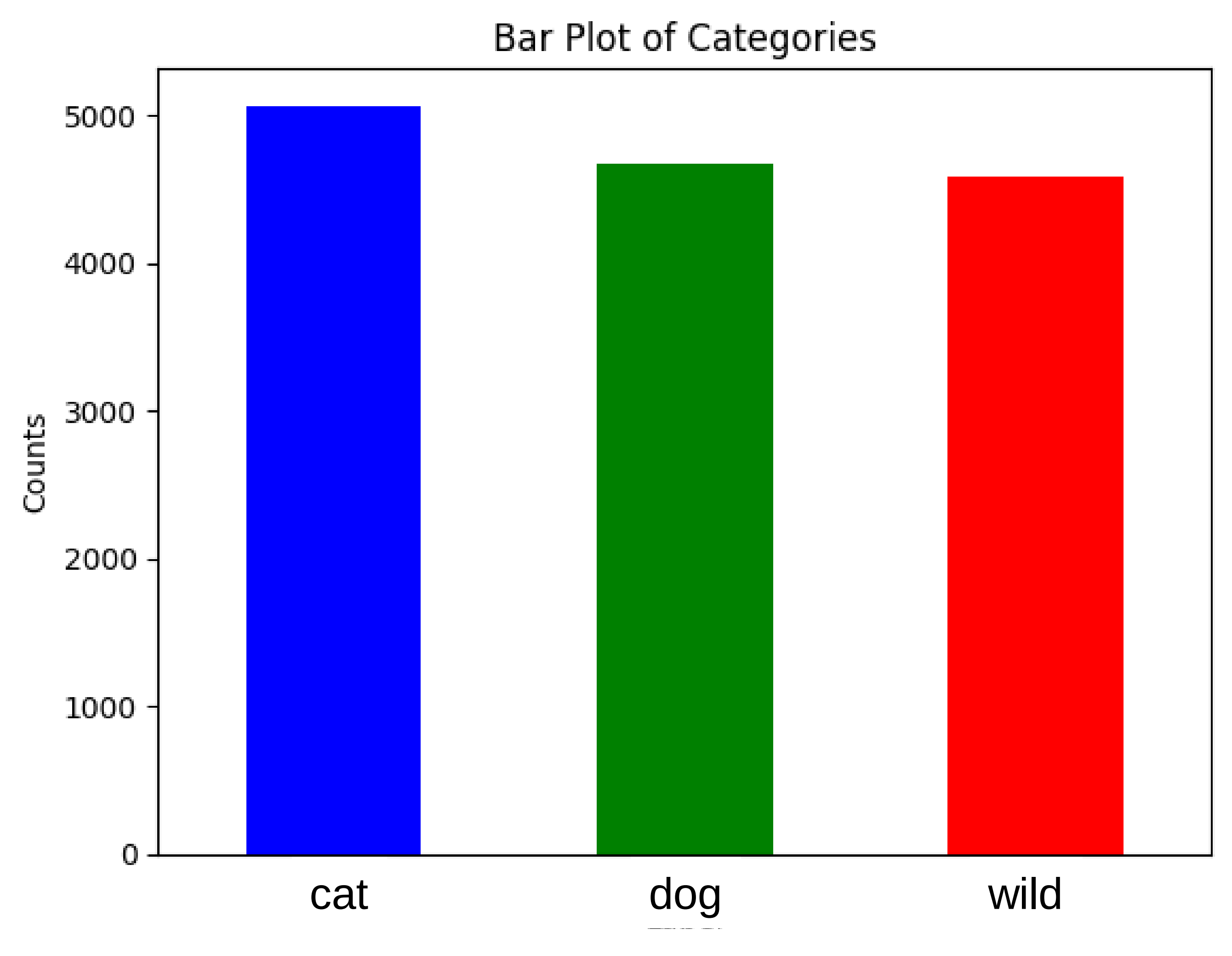
Exploratory Data Analysis

- Dataset is hosted on Kaggle
  https://www.kaggle.com/datasets/andrewmvd/animal-faces/data
- Pictures are organized into folders by class.
- Pictures already split between Test and Train sets.

EDA - Examples



EDA - Distribution



Preprocessing

- Images dont need a whole lot of preprocessing and just need to apply an image generator. Images are clean and formated well but looking at the sample images above the images look very similar in layout and face size. Lets use imagedatagenerator to introduce some variability to the dataset. Hopefully this will allow the model to extract better features.

```
batch = 32
train_datagen = ImageDataGenerator(
    rotation_range=45,
    width_shift_range=0.1,
    brightness_range=[0.6,1.4],
    height_shift_range=0.1,
    shear_range=0.2,
    zoom_range=0.1,
    horizontal_flip=True,
    fill_mode='nearest')

train_generator1 = train_datagen.flow_from_directory(
    directory=train_path,
    target_size=(512, 512),
    color_mode="rgb",
    batch_size=batch,
    class_mode="categorical",
    shuffle=True)
```

Model Design

- For the model I will be using a convolutional neural network (CNN) using Keras. This model will have an initial layout of 3 convolution layers for feature extraction, 1 dense layer, and a final output layer.
- I built three models, an initial basic model with no image augmentation.
- A 2^{nd} basic model using image augmentation
- And a third final model after tuning parameters.

Initial Model – no preprocessing

```
# parameters
kernel_size = (3,3)
pool_size= (2,2)
conv1 = 16
conv2 = 32
conv3 = 64
dense = 32
dropout_dense = 0.5
epoch = 5
rate = .0005
```

- 3 Convolution layers
  - Relu activation
  - Max pooling
- 1 dense layer
- Final 3 output layer

```
# Train the initial model
mod_initial = model.fit(X_train, y_train, batch_size=batch, epochs = epoch)
```

```
Epoch 1/5
448/448 ━━━━━━━━━━━━━━━━━━━━ 59s 105ms/step - accuracy: 0.3565 - loss: 40.9473
Epoch 2/5
448/448 ━━━━━━━━━━━━━━━━━━━━ 49s 108ms/step - accuracy: 0.3550 - loss: 1.0978
Epoch 3/5
448/448 ━━━━━━━━━━━━━━━━━━━━ 48s 108ms/step - accuracy: 0.3575 - loss: 1.0974
Epoch 4/5
448/448 ━━━━━━━━━━━━━━━━━━━━ 48s 108ms/step - accuracy: 0.3543 - loss: 1.0977
Epoch 5/5
448/448 ━━━━━━━━━━━━━━━━━━━━ 48s 108ms/step - accuracy: 0.3535 - loss: 1.0977
```

Model 2 - DataGen

- Same initial model using augmented image data using preprocessed image generator

```
# model fit with image generator
mod_generator1 = model.fit(train_generator1, epochs=epoch)
```

```
Epoch 1/5
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_d
  self._warn_if_super_not_called()
448/448 ━━━━━━━━━━━━━━━━━━━━ 969s 2s/step - accuracy: 0.3491 - loss: 1.0981
Epoch 2/5
448/448 ━━━━━━━━━━━━━━━━━━━━ 951s 2s/step - accuracy: 0.3534 - loss: 1.0976
Epoch 3/5
448/448 ━━━━━━━━━━━━━━━━━━━━ 945s 2s/step - accuracy: 0.3519 - loss: 1.0979
Epoch 4/5
448/448 ━━━━━━━━━━━━━━━━━━━━ 939s 2s/step - accuracy: 0.3517 - loss: 1.0978
Epoch 5/5
448/448 ━━━━━━━━━━━━━━━━━━━━ 951s 2s/step - accuracy: 0.3539 - loss: 1.0977
```

Tuning

- Now on to tuning the model. I approached tuning from several different areas.
- First parameters for Data Augmentation
- Second parameters for each Convolution layer(kernel size, stride, pool size).
- Third the dimensions and depth of the Convolution layers.
- And last the size and parameters for the dense layer.

Model 3 Final – parameters

- Parameters – Changed Kernel size, filters, stride, etc.
- Data augmentation – Tried with and without
- Convolution layers – tried more width(filters) and more depth(layers), activation functions, dropout, etc.
- Dense layer – changed width and depth
- Training – Tried different optimizers, Epoch, learning rate, etc.

Model 3 – Final Conv. Layers

- 5 Convolution Layers Total
- Input shape of 512x512x3
- Output shape to dense layer of 16x16x1024
- Each layer same architecture but different parameters.

Layer 1

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 506, 506, 128)	18,944
leaky_re_lu (LeakyReLU)	(None, 506, 506, 128)	0
max_pooling2d (MaxPooling2D)	(None, 253, 253, 128)	0
dropout (Dropout)	(None, 253, 253, 128)	0

*
*
* Layer 5

conv2d_4 (Conv2D)	(None, 32, 32, 1024)	4,719,616
leaky_re_lu_4 (LeakyReLU)	(None, 32, 32, 1024)	0
max_pooling2d_4 (MaxPooling2D)	(None, 16, 16, 1024)	0
dropout_4 (Dropout)	(None, 16, 16, 1024)	0

Model 3 Final – Dense Layer

- 64 Neuron dense layer
- Output layer with 3 outputs
- Relu Activation

flatten (Flatten)	(None, 262144)	0
dense (Dense)	(None, 64)	16,777,216
activation (Activation)	(None, 64)	0
dropout_5 (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 3)	195

```
Total params: 24,990,531 (95.33 MB)
Trainable params: 24,990,531 (95.33 MB)
Non-trainable params: 0 (0.00 B)
```

Model 3 Final – What didn’t work.

- Biggest Issue was getting model to converge and training time.
- Overfitted with too much complexity. > 30 million parameters.
- Increasing learning rate
- Adding more wider and deeper dense layers.
- Image augmentation

Model 3 Final – What did work.

- Learning Rate with Momentum. Rate = .00001

```
model2.compile(
    optimizer=SGD(learning_rate=rate, momentum=0.9),
    loss='categorical_crossentropy',
    metrics=['accuracy'])
```

- Decreasing Kernel size with Increasing filters.

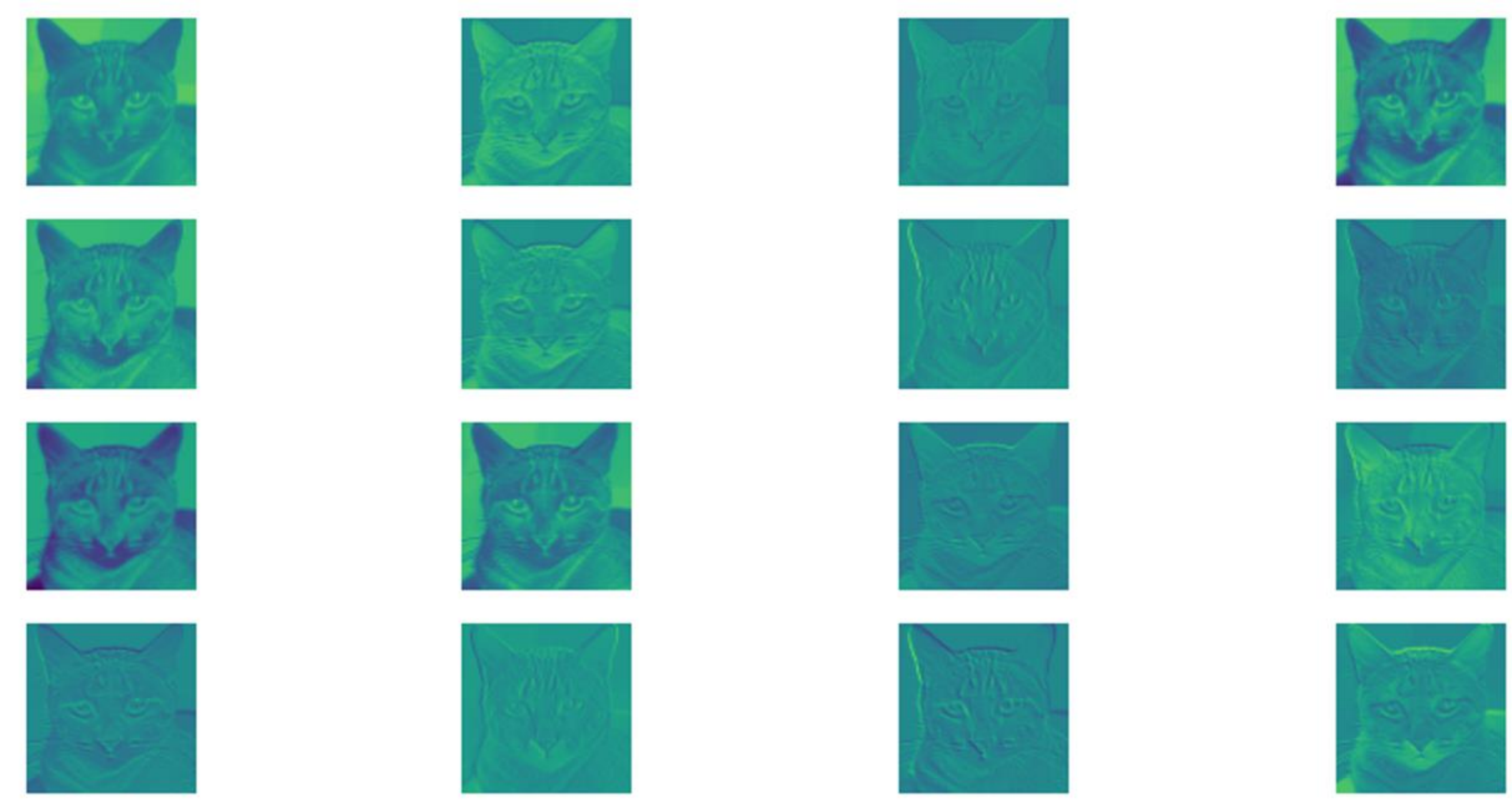
```
kernel1 = 7
kernel2 = 5
kernel3 = 3
```

```
conv1 = 128
conv2 = 256
conv3 = 384
conv4 = 512
conv5 = 1024
```

- More Epochs and Early stopping
- Medium Complexity

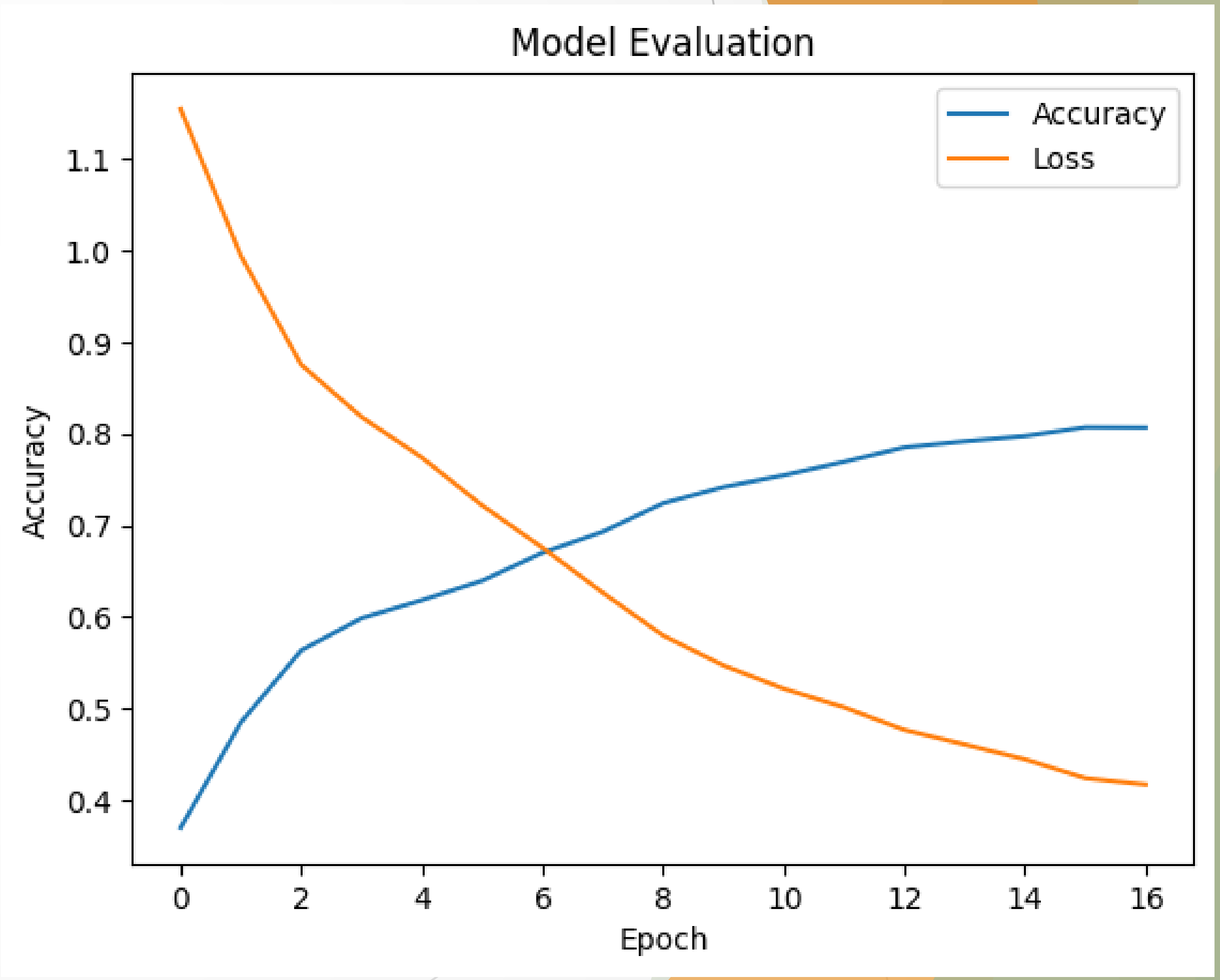
```
Total params: 24,990,531 (95.33 MB)
Trainable params: 24,990,531 (95.33 MB)
Non-trainable params: 0 (0.00 B)
```

Visualizing Convolution Filters



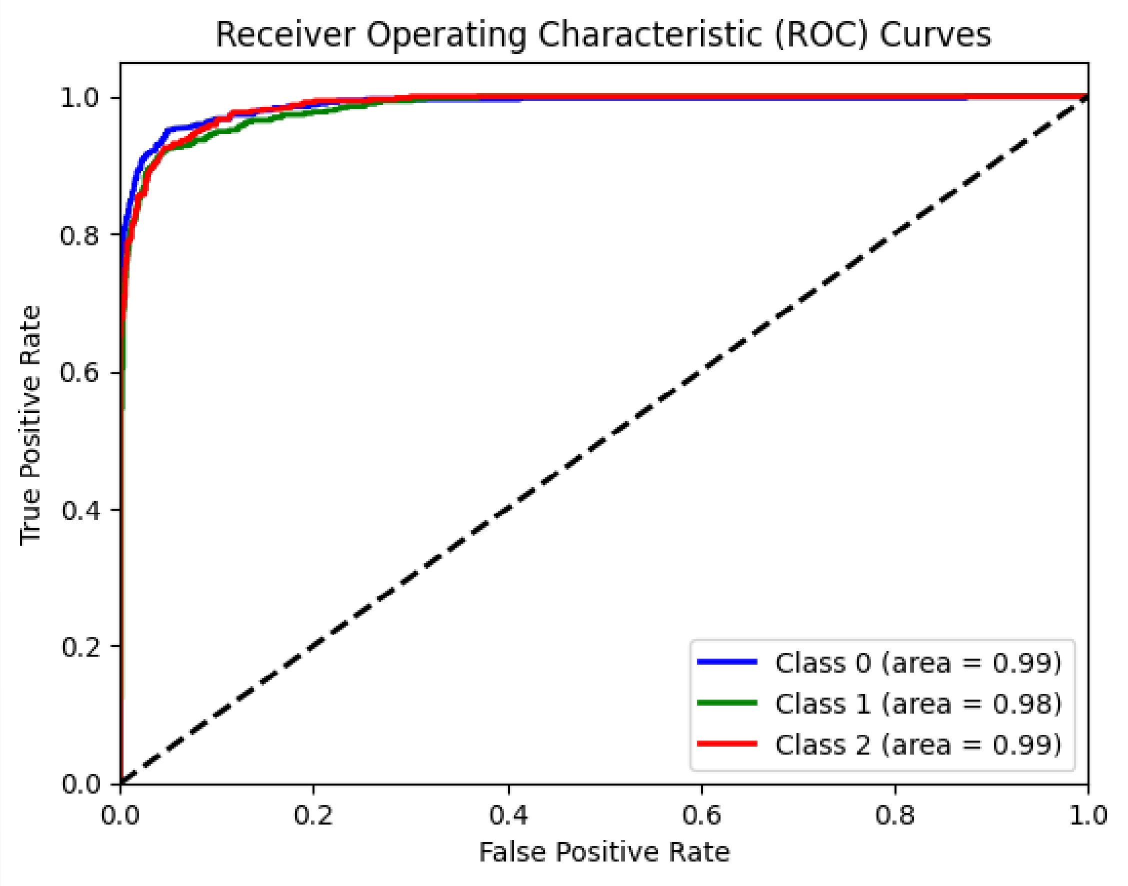
Final Results – Training

```
Epoch 1/25
1792/1792 ━━━━━━━━━━━━━━━━━━━━ 142s 64ms/step - accuracy: 0.3558 - loss: 1.4748
Epoch 2/25
1792/1792 ━━━━━━━━━━━━━━━━━━━━ 115s 64ms/step - accuracy: 0.4598 - loss: 1.0254
Epoch 3/25
1792/1792 ━━━━━━━━━━━━━━━━━━━━ 115s 64ms/step - accuracy: 0.5548 - loss: 0.8937
Epoch 4/25
1792/1792 ━━━━━━━━━━━━━━━━━━━━ 115s 64ms/step - accuracy: 0.5937 - loss: 0.8223
Epoch 5/25
1792/1792 ━━━━━━━━━━━━━━━━━━━━ 115s 64ms/step - accuracy: 0.6081 - loss: 0.7886
Epoch 6/25
1792/1792 ━━━━━━━━━━━━━━━━━━━━ 115s 64ms/step - accuracy: 0.6278 - loss: 0.7364
Epoch 7/25
1792/1792 ━━━━━━━━━━━━━━━━━━━━ 115s 64ms/step - accuracy: 0.6627 - loss: 0.6886
Epoch 8/25
1792/1792 ━━━━━━━━━━━━━━━━━━━━ 115s 64ms/step - accuracy: 0.6891 - loss: 0.6345
Epoch 9/25
1792/1792 ━━━━━━━━━━━━━━━━━━━━ 115s 64ms/step - accuracy: 0.7162 - loss: 0.5930
Epoch 10/25
1792/1792 ━━━━━━━━━━━━━━━━━━━━ 115s 64ms/step - accuracy: 0.7328 - loss: 0.5617
Epoch 11/25
1792/1792 ━━━━━━━━━━━━━━━━━━━━ 115s 64ms/step - accuracy: 0.7483 - loss: 0.5270
Epoch 12/25
1792/1792 ━━━━━━━━━━━━━━━━━━━━ 115s 64ms/step - accuracy: 0.7630 - loss: 0.5048
Epoch 13/25
1792/1792 ━━━━━━━━━━━━━━━━━━━━ 115s 64ms/step - accuracy: 0.7824 - loss: 0.4822
Epoch 14/25
1792/1792 ━━━━━━━━━━━━━━━━━━━━ 115s 64ms/step - accuracy: 0.7934 - loss: 0.4562
Epoch 15/25
1792/1792 ━━━━━━━━━━━━━━━━━━━━ 115s 64ms/step - accuracy: 0.7960 - loss: 0.4438
Epoch 16/25
1792/1792 ━━━━━━━━━━━━━━━━━━━━ 115s 64ms/step - accuracy: 0.8077 - loss: 0.4205
Epoch 17/25
1792/1792 ━━━━━━━━━━━━━━━━━━━━ 115s 64ms/step - accuracy: 0.8062 - loss: 0.4178
Epoch 17: early stopping
```



Final Results – Test

```
46/46 ━━━━━━━━━━━━━━━━━━━━ 5s 85ms/step - accuracy: 0.9287 - loss: 0.2152
Test Accuracy: 0.91
Test Loss: 0.25
```



Conclusion