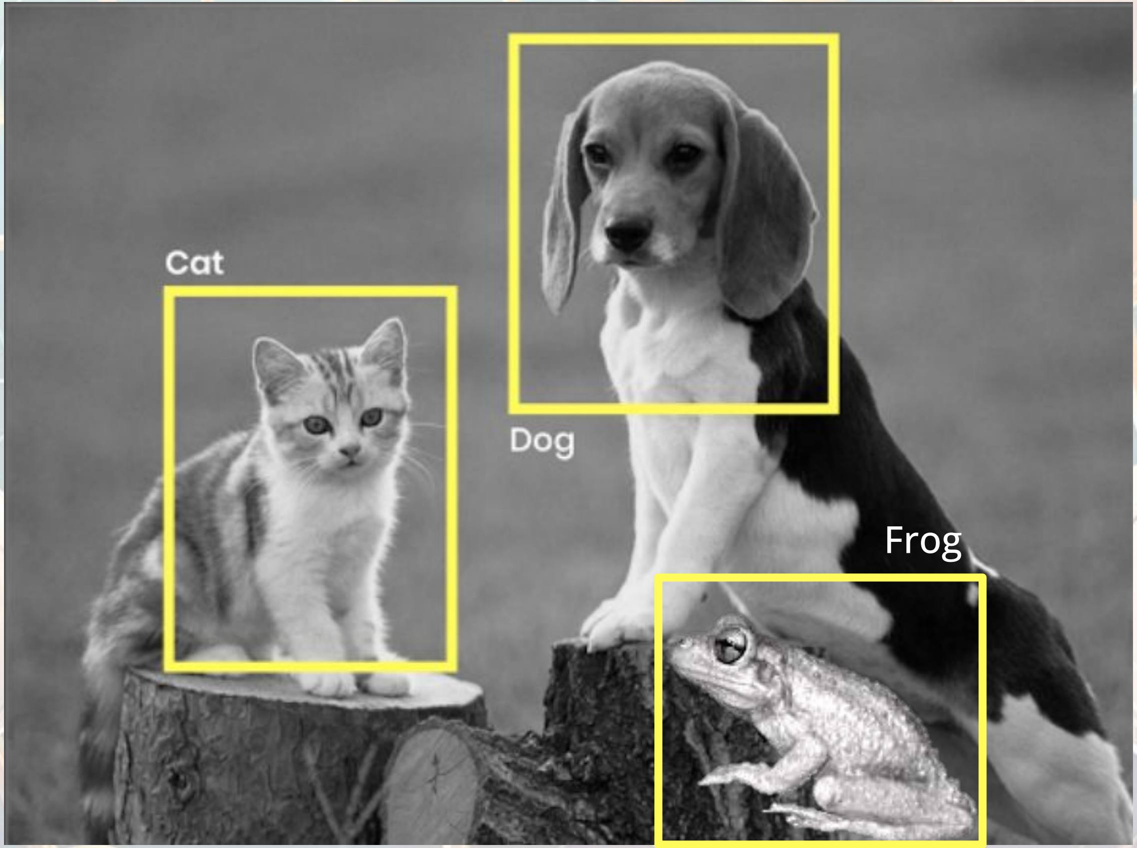
Neural Network Image Classification

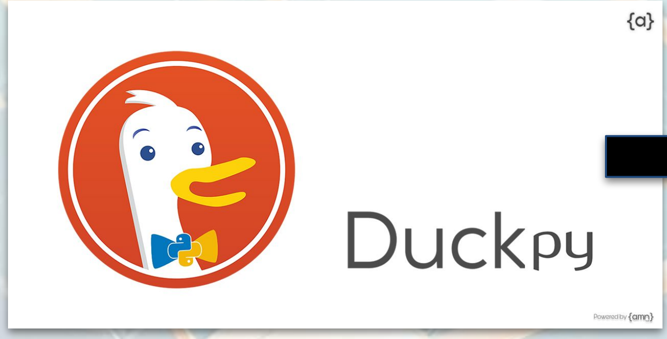
2023 October 28th, ALY6020

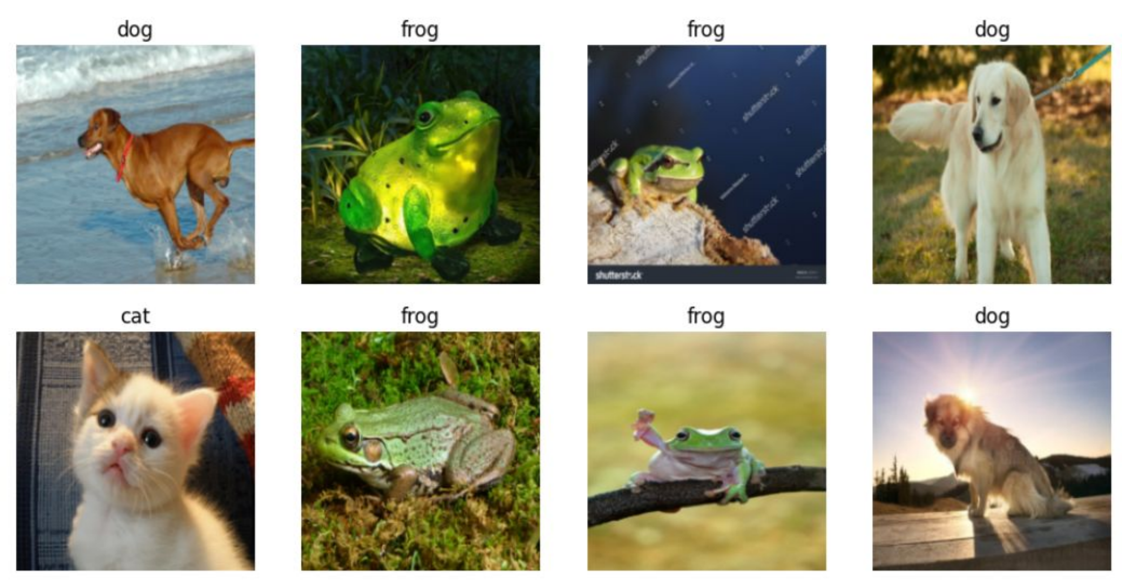
Presented to Professor Behzad Ahmadi
Prepared by Heejae Roh

Image Classification with Neural Network



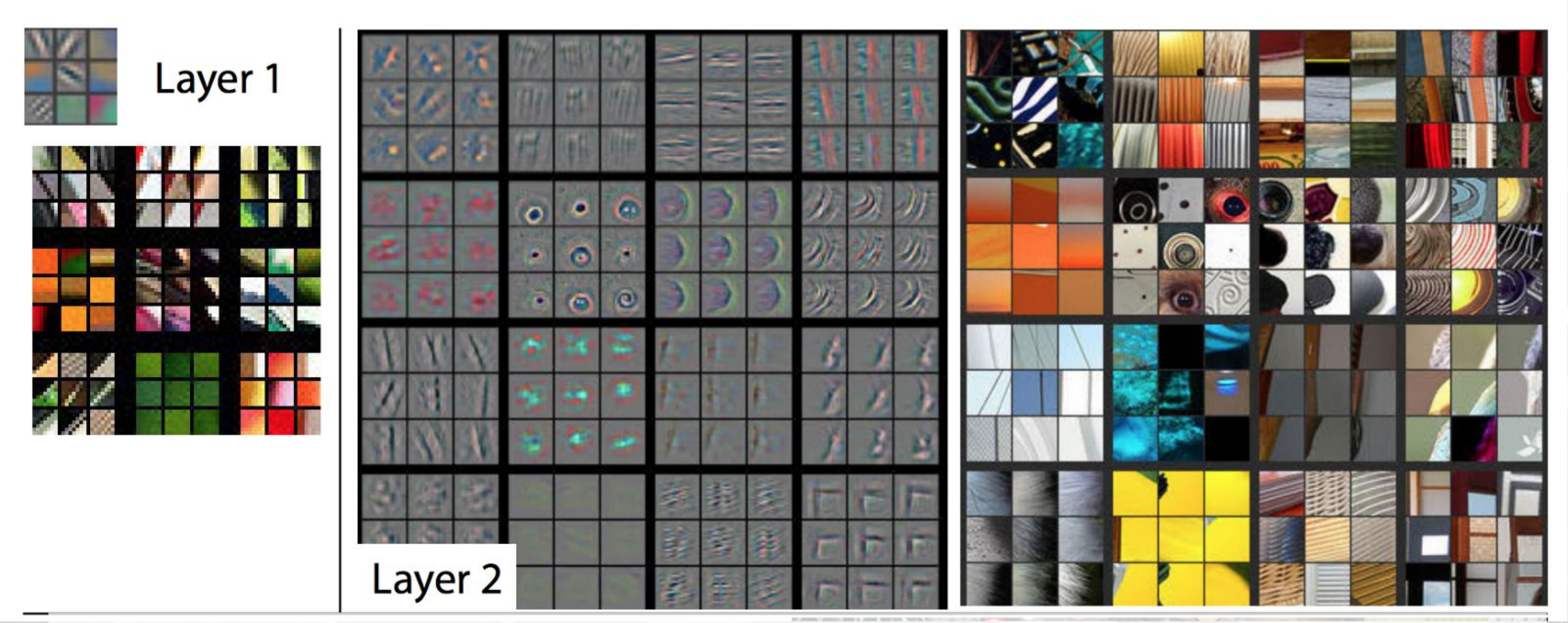
Duckduckgo Search in python
: Cat, Dog, and Frog





- ‘cat’: 872, ‘dog’: 867, ‘frog’: 876… 115 images removed due to verification failure
- Total number of images: 2,484
- Training images: 1,988
- Validation Images (20%): 496

Use fast.ai library

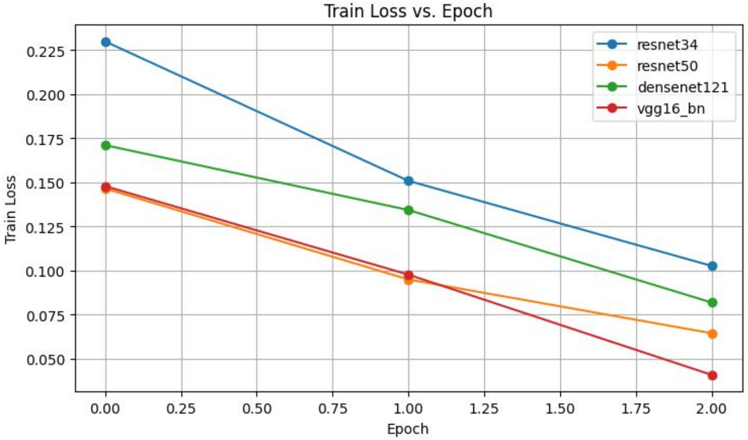


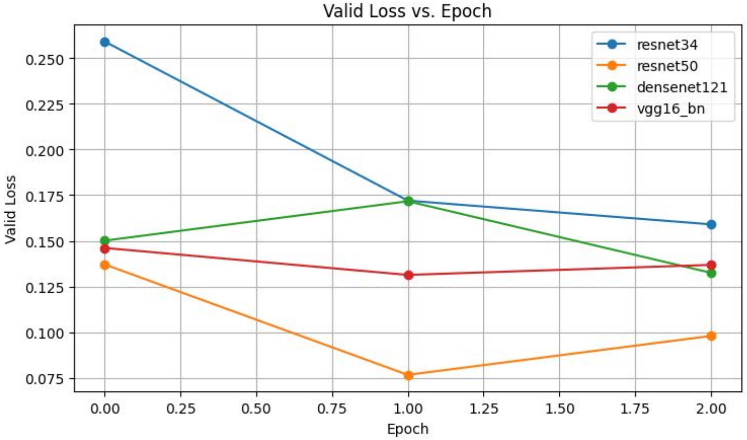
- A deep learning library
- all the necessary functions
  to define a Dataset and train a model for computer vision tasks

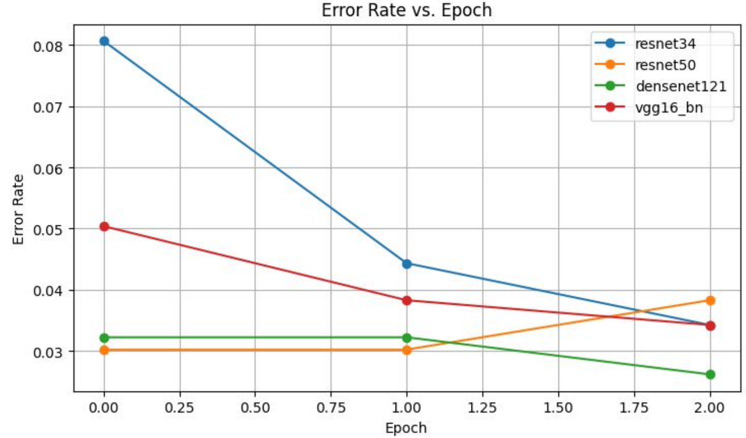
Data Preparation

- Removing due to verification failure
- Using ‘verify_images’ function in fastai.vision.utils.

Checking various models

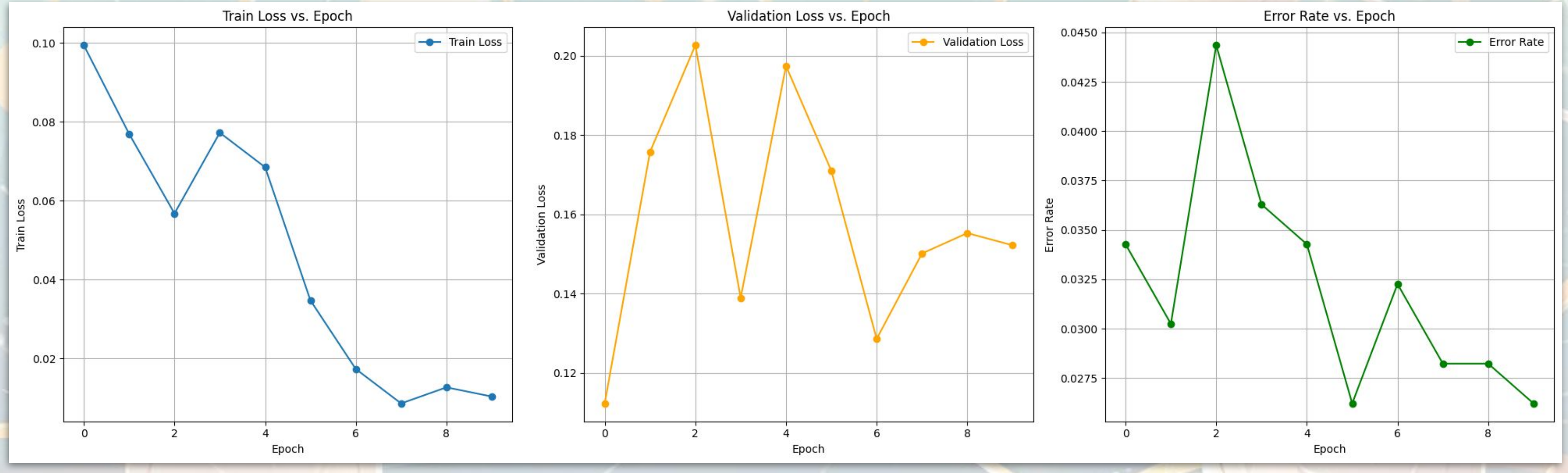






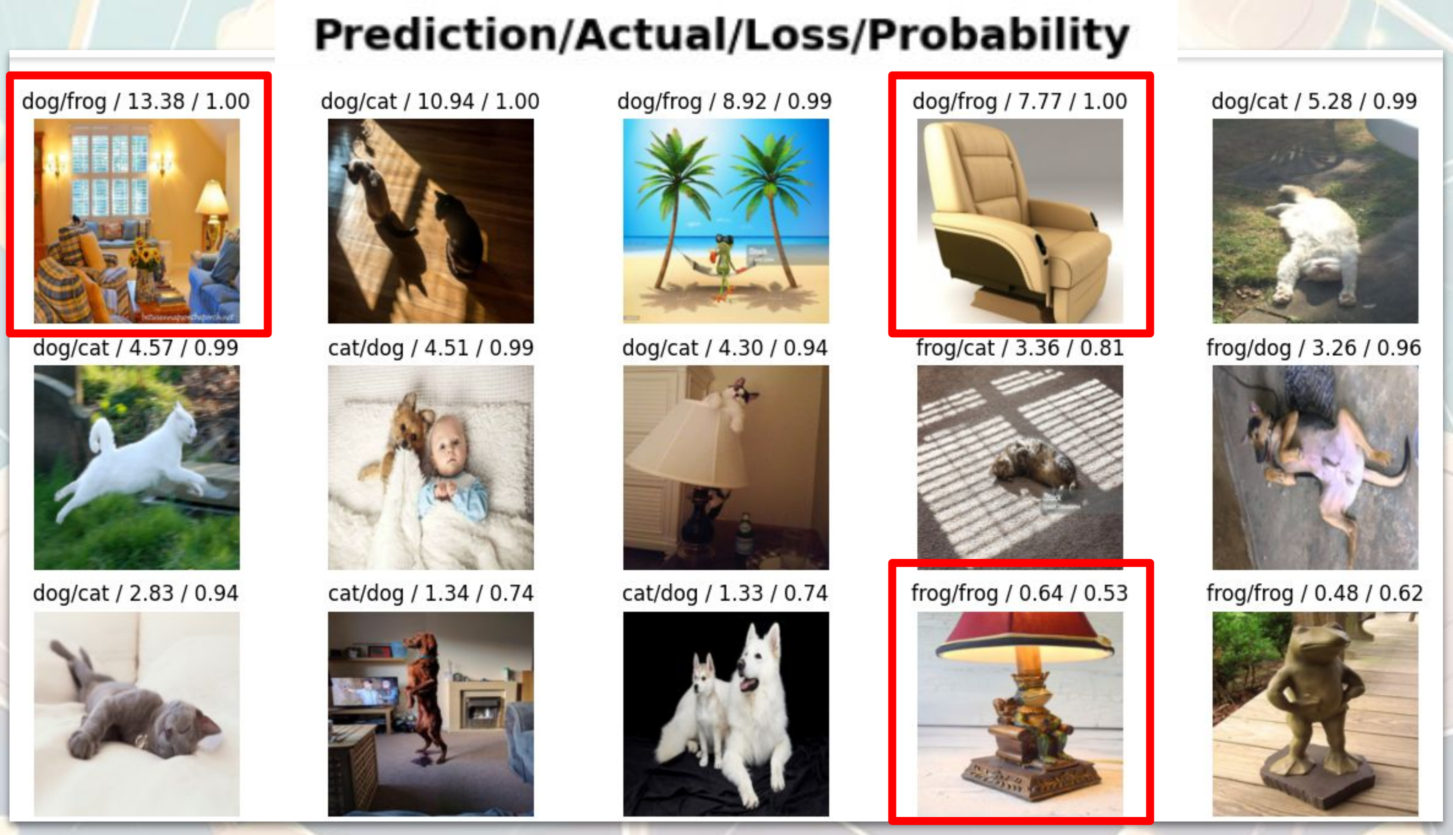
- ‘densenet121’, the model with the lowest error_rate.
- Among [resnet18, resnet34, resnet50, densenet121, vgg16_bn]

Train, Valid Loss & Error with densenet121, epoch=10



- Error_rate is 2.6%, in other words, accuracy is 97.4% in validation dataset
- Train_loss getting lower and lower as epoch increases
- Validation_loss not getting lower after 6 epoch
- Error_rate getting lower and lower as epochs increases

Mispredicted images with densenet121, epoch=10



Prediction/Actual/Loss/Probability

dog/frog / 13.38 / 1.00

dog/cat / 10.94 / 1.00

dog/frog / 8.92 / 0.99

dog/frog / 7.77 / 1.00

dog/cat / 5.28 / 0.99

dog/cat / 4.57 / 0.99

cat/dog / 4.51 / 0.99

dog/cat / 4.30 / 0.94

frog/cat / 3.36 / 0.81

frog/dog / 3.26 / 0.96

dog/cat / 2.83 / 0.94

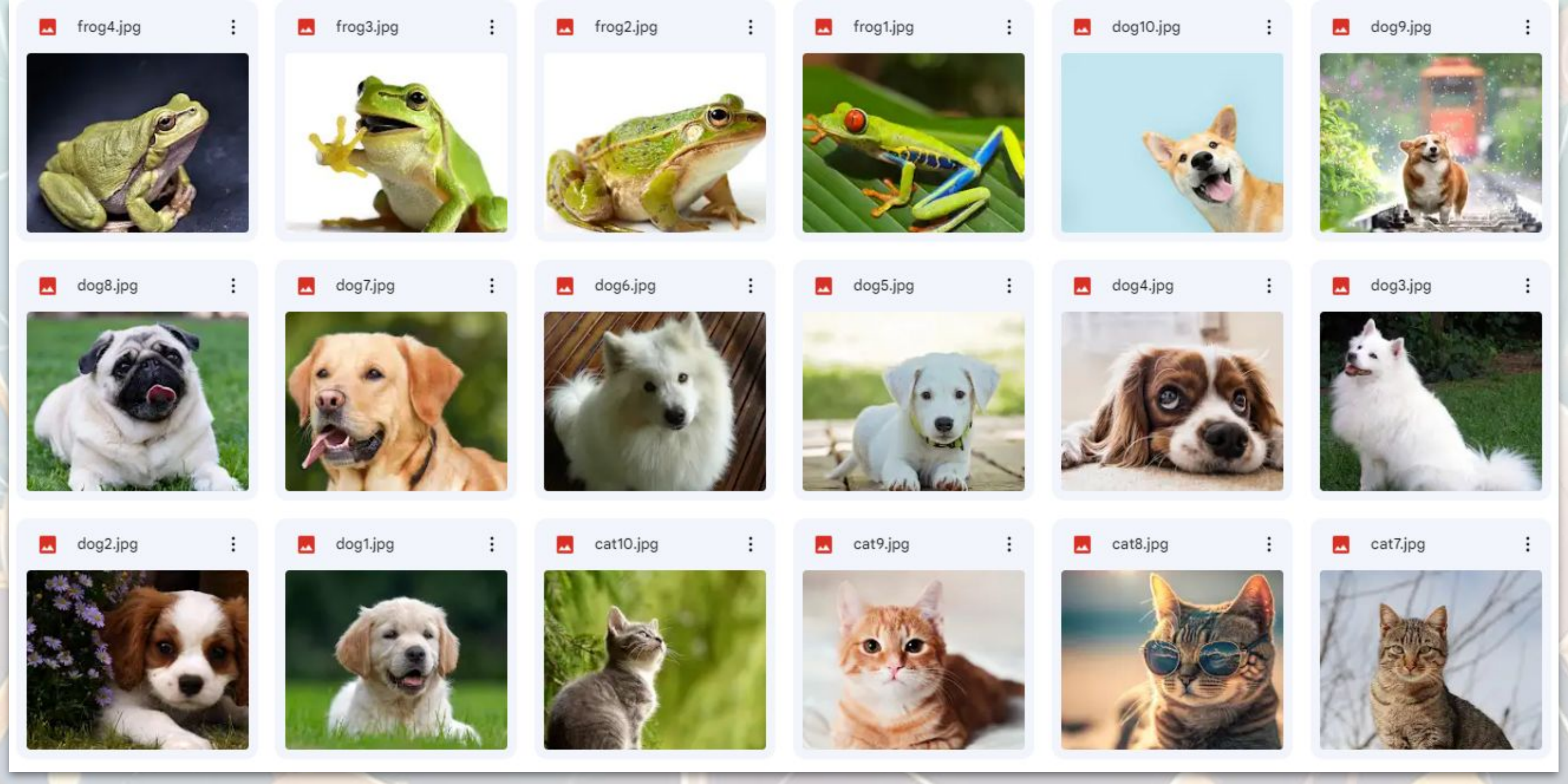
cat/dog / 1.34 / 0.74

cat/dog / 1.33 / 0.74

frog/frog / 0.64 / 0.53

frog/frog / 0.48 / 0.62

Predict.model with new test dataset in google Drive



Predict.model with new test dataset

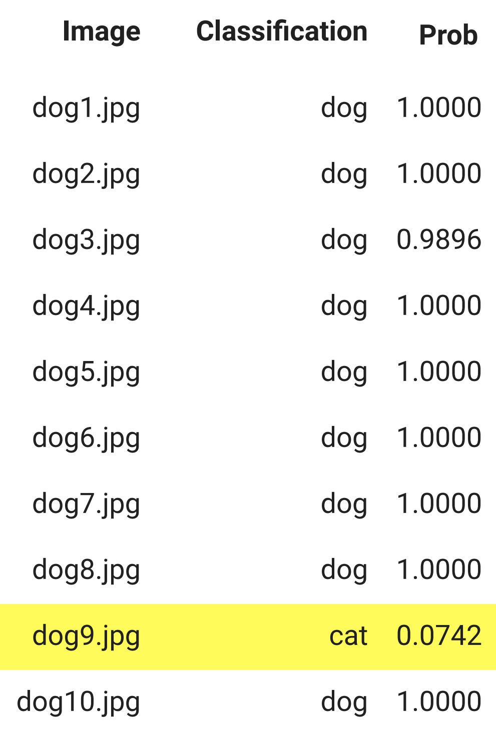


Image	Classification	Prob
dog1.jpg	dog	1.0000
dog2.jpg	dog	1.0000
dog3.jpg	dog	0.9896
dog4.jpg	dog	1.0000
dog5.jpg	dog	1.0000
dog6.jpg	dog	1.0000
dog7.jpg	dog	1.0000
dog8.jpg	dog	1.0000
dog9.jpg	cat	0.0742
dog10.jpg	dog	1.0000

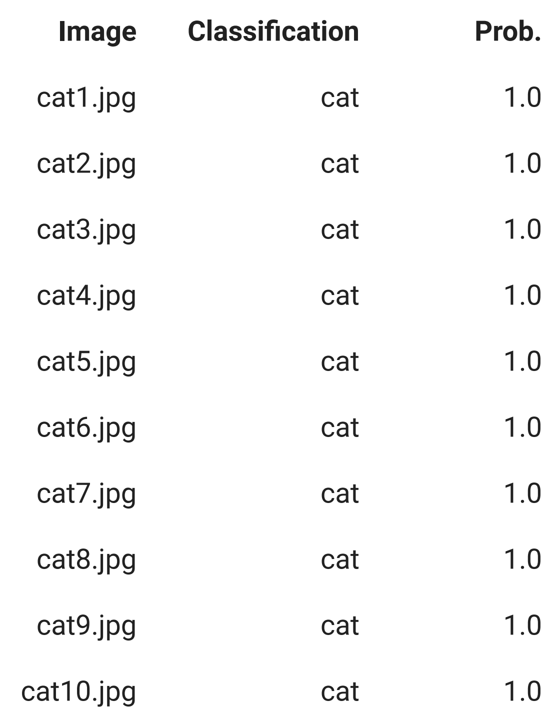


Image	Classification	Prob.
cat1.jpg	cat	1.0
cat2.jpg	cat	1.0
cat3.jpg	cat	1.0
cat4.jpg	cat	1.0
cat5.jpg	cat	1.0
cat6.jpg	cat	1.0
cat7.jpg	cat	1.0
cat8.jpg	cat	1.0
cat9.jpg	cat	1.0
cat10.jpg	cat	1.0

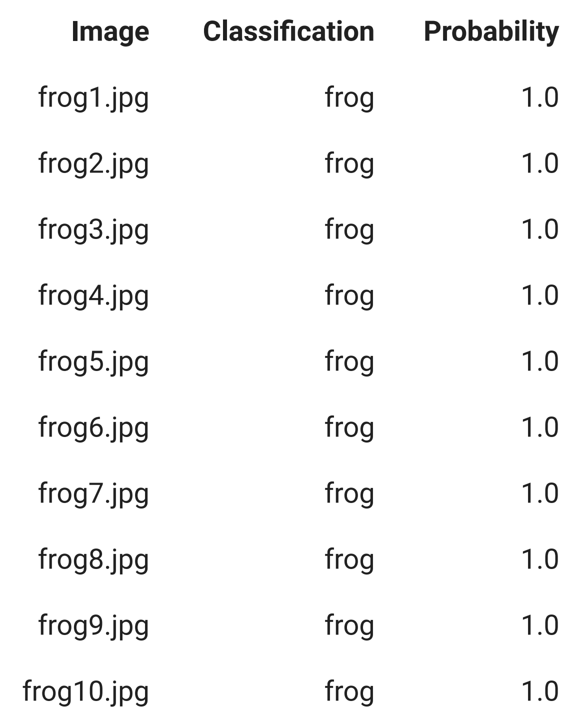


Image	Classification	Probability
frog1.jpg	frog	1.0
frog2.jpg	frog	1.0
frog3.jpg	frog	1.0
frog4.jpg	frog	1.0
frog5.jpg	frog	1.0
frog6.jpg	frog	1.0
frog7.jpg	frog	1.0
frog8.jpg	frog	1.0
frog9.jpg	frog	1.0
frog10.jpg	frog	1.0

- Only dog9.jpg is mispredicted as cat

dog9.jpg

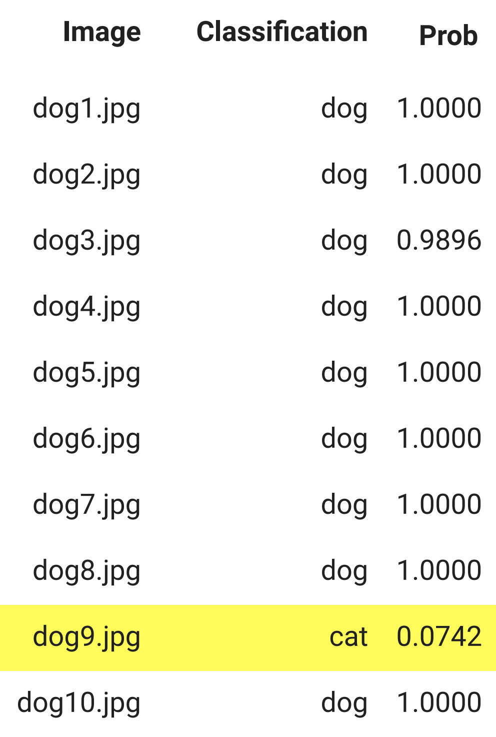
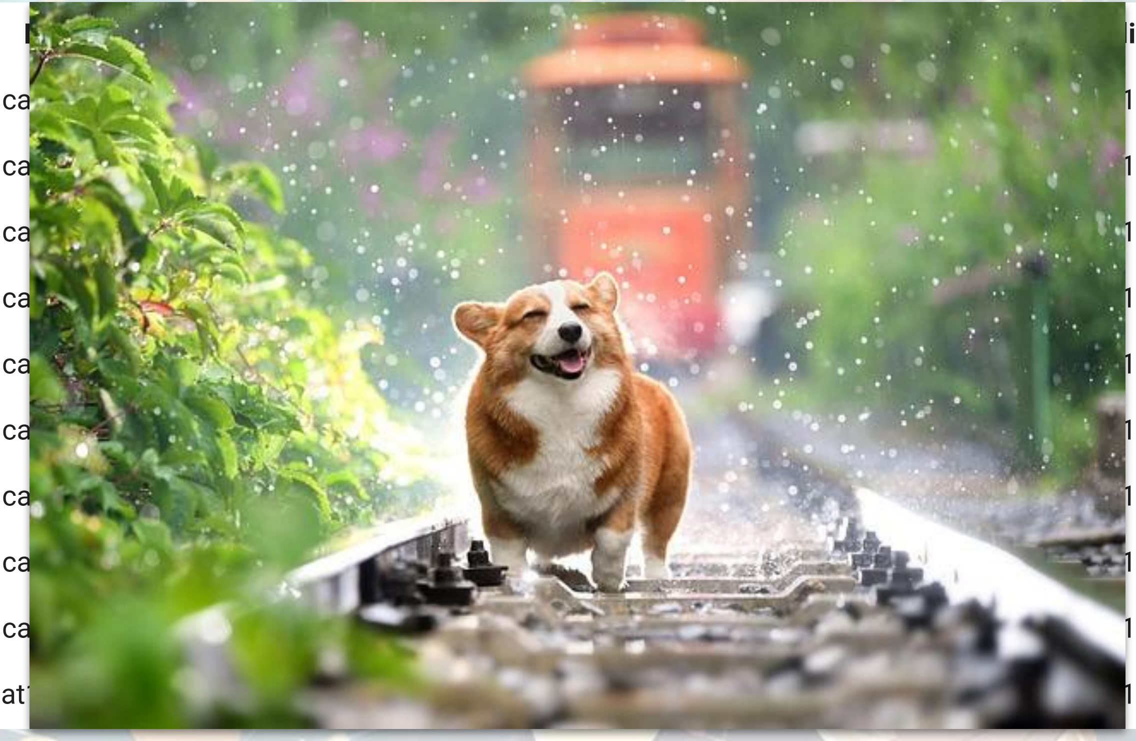


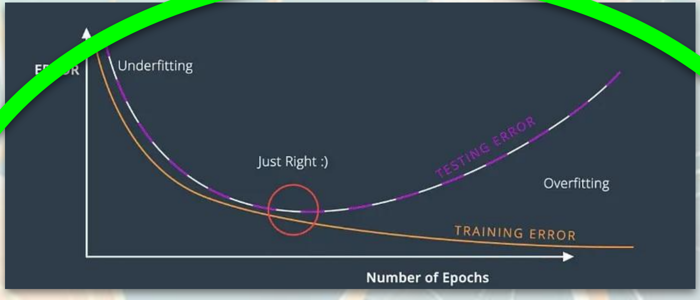
Image	Classification	Prob
dog1.jpg	dog	1.0000
dog2.jpg	dog	1.0000
dog3.jpg	dog	0.9896
dog4.jpg	dog	1.0000
dog5.jpg	dog	1.0000
dog6.jpg	dog	1.0000
dog7.jpg	dog	1.0000
dog8.jpg	dog	1.0000
dog9.jpg	cat	0.0742
dog10.jpg	dog	1.0000

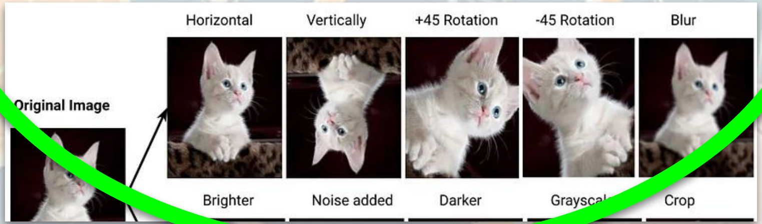
	ity
ca	1.0
ca	1.0
ca	1.0
ca	1.0
ca	1.0
ca	1.0
ca	1.0
ca	1.0
ca	1.0
cat1	1.0



- Only dog9.jpg is mispredicted as cat
- This is thought to be confused with a cat because dog9 has pointed triangular ears

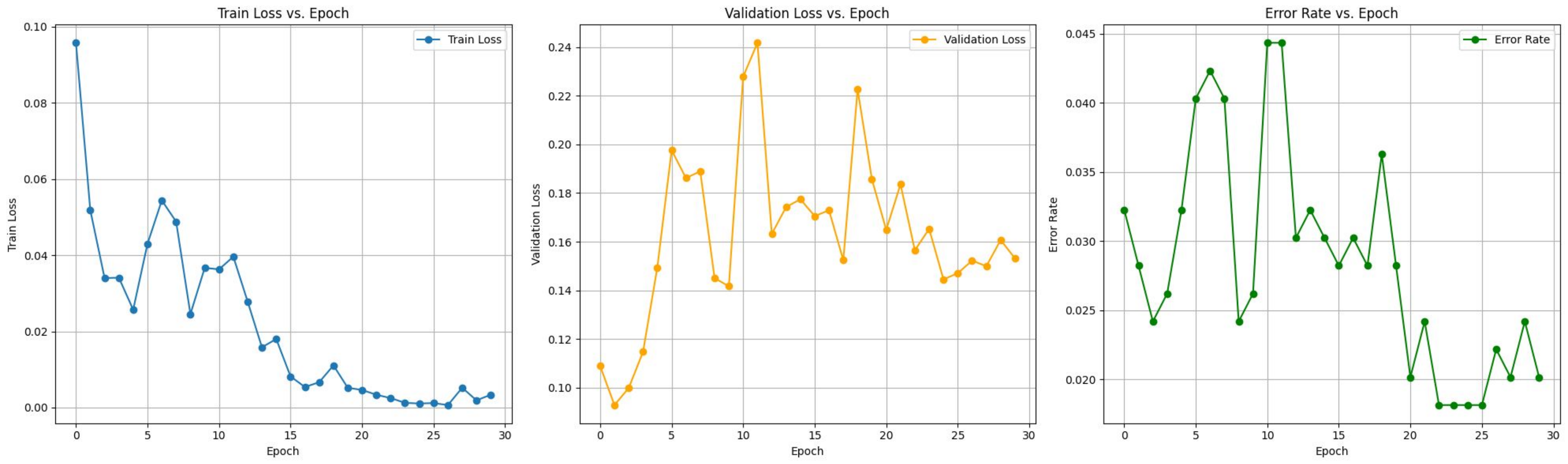
How to make dog9.jpg as dog





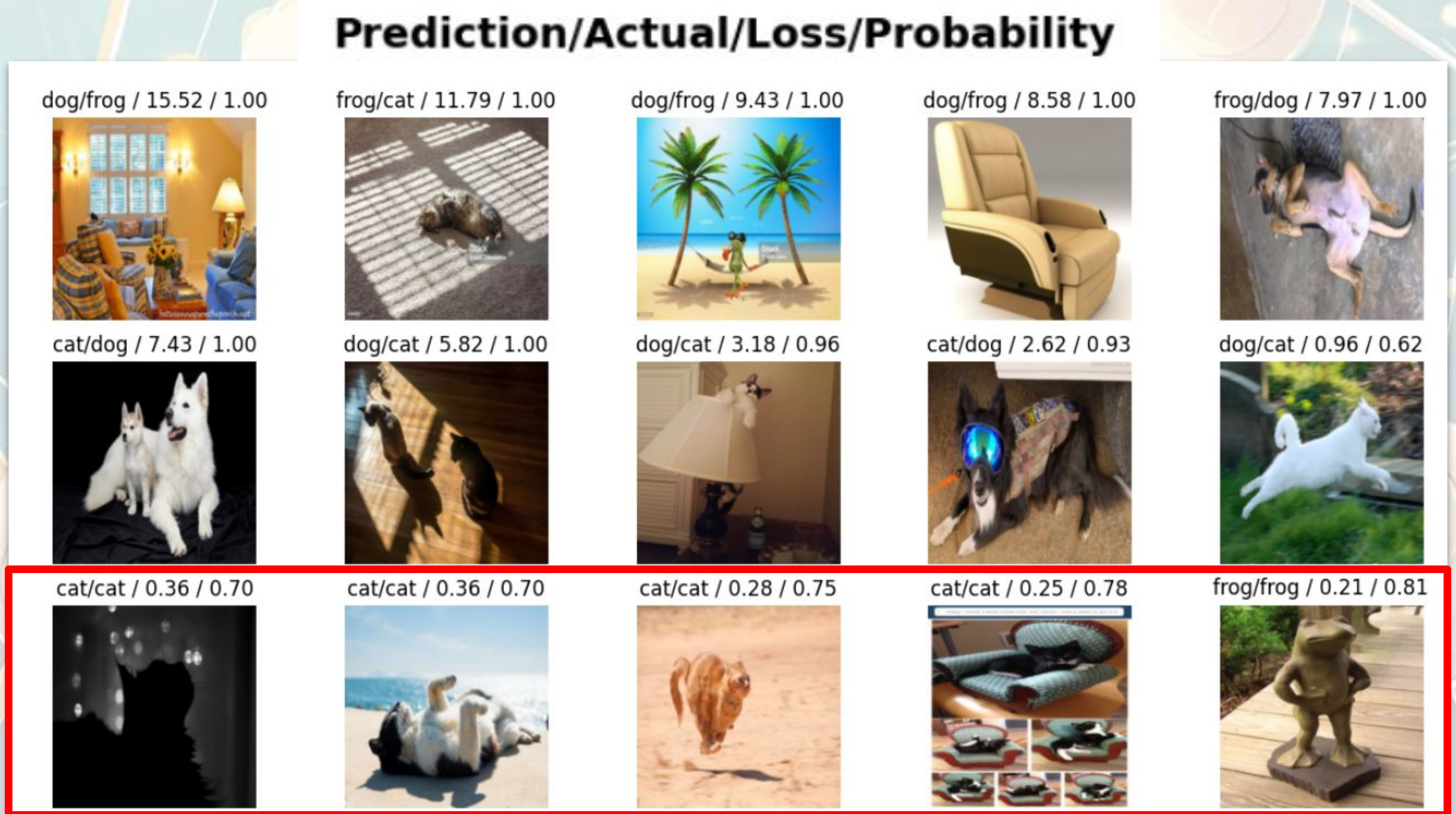
- Improvements can be made by feeding more photos of dogs with pointy ears to the model.
- However, in this case, let’s assume that we have limited train data, I can only modify the model to predict dog9 as dog.

Train, Valid Loss & Error with densenet121, epoch=30



- Error_rate is 2.0%, in other words, accuracy is 98.0% in validation dataset
- Train_loss getting lower and lower as epoch increases
- Validation_loss not getting lower after 10 epoch
- Error_rate getting lower and lower as epochs increases

Mispredicted images with densenet121, epoch=30



Prediction/Actual/Loss/Probability

Predict.model with test dataset

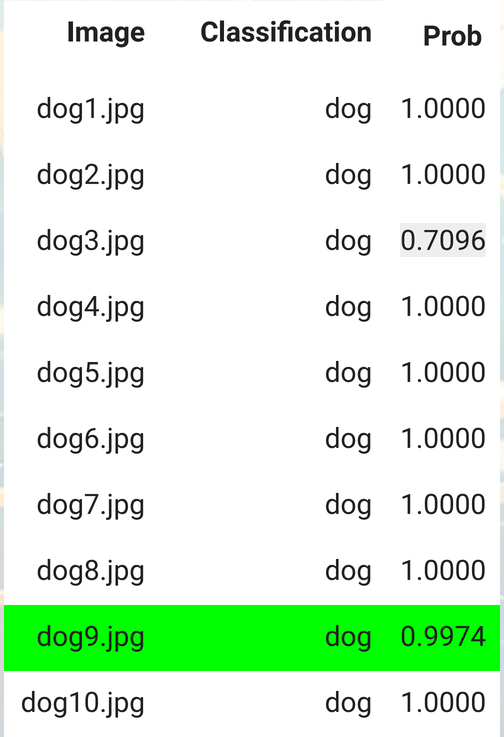


Image	Classification	Prob
dog1.jpg	dog	1.0000
dog2.jpg	dog	1.0000
dog3.jpg	dog	0.7096
dog4.jpg	dog	1.0000
dog5.jpg	dog	1.0000
dog6.jpg	dog	1.0000
dog7.jpg	dog	1.0000
dog8.jpg	dog	1.0000
dog9.jpg	dog	0.9974
dog10.jpg	dog	1.0000

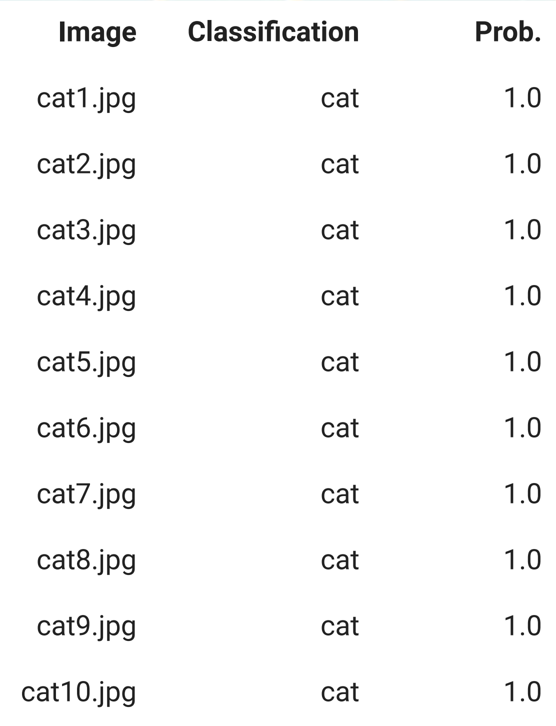


Image	Classification	Prob.
cat1.jpg	cat	1.0
cat2.jpg	cat	1.0
cat3.jpg	cat	1.0
cat4.jpg	cat	1.0
cat5.jpg	cat	1.0
cat6.jpg	cat	1.0
cat7.jpg	cat	1.0
cat8.jpg	cat	1.0
cat9.jpg	cat	1.0
cat10.jpg	cat	1.0

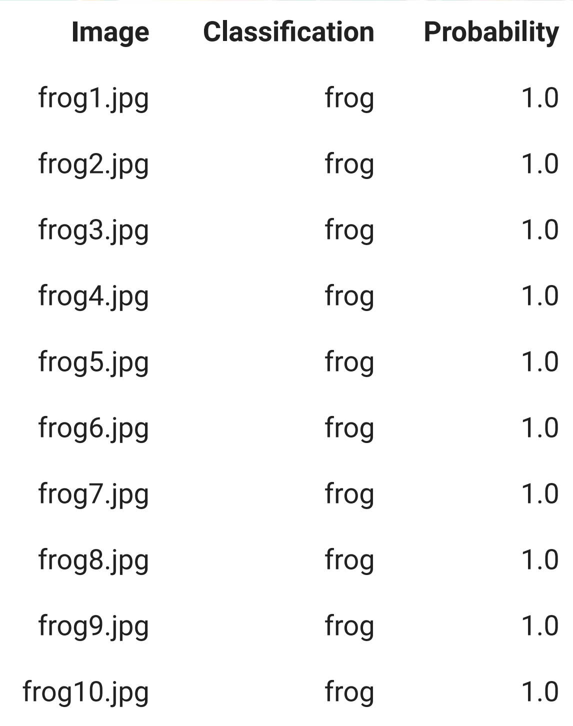
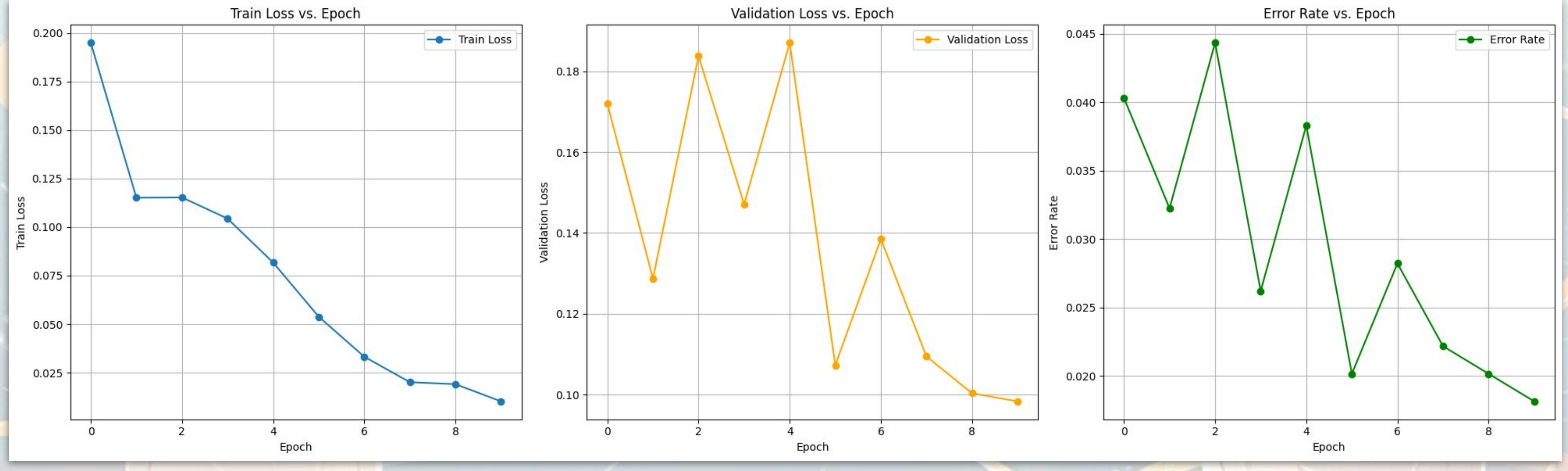


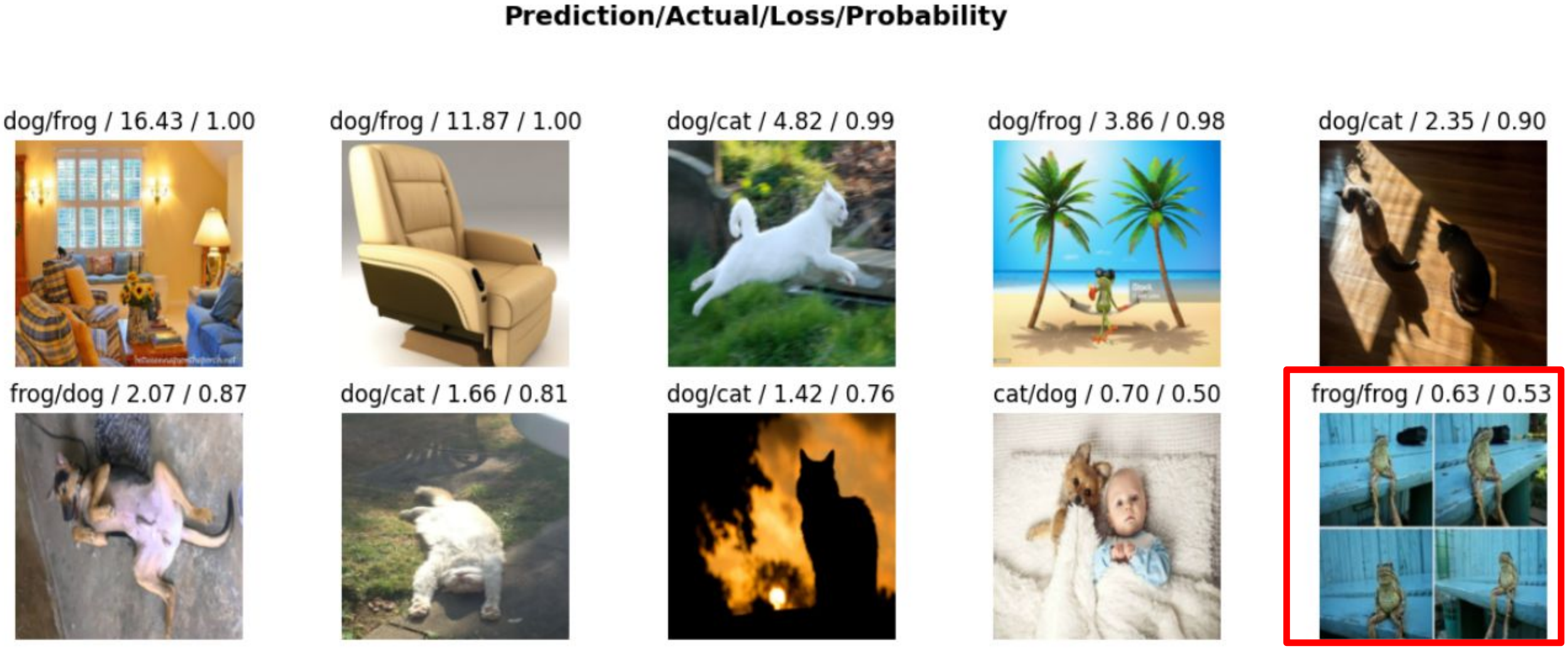
Image	Classification	Probability
frog1.jpg	frog	1.0
frog2.jpg	frog	1.0
frog3.jpg	frog	1.0
frog4.jpg	frog	1.0
frog5.jpg	frog	1.0
frog6.jpg	frog	1.0
frog7.jpg	frog	1.0
frog8.jpg	frog	1.0
frog9.jpg	frog	1.0
frog10.jpg	frog	1.0

Train, Valid Loss & Error with densenet121, epoch=10, with augmentation



- Error_rate is 1.8%, in other words, accuracy is 98.2% in validation dataset
- Train_loss getting lower and lower as epoch increases
- Validation_loss getting lower and lower as epoch increases
- Error_rate getting lower and lower as epochs increases

Mispredicted images with densenet121, epoch=10, with augmentation



Predict.model with test dataset

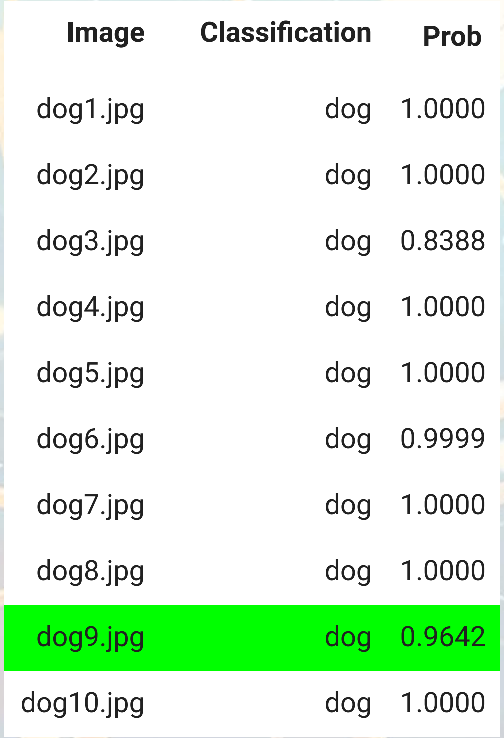


Image	Classification	Prob
dog1.jpg	dog	1.0000
dog2.jpg	dog	1.0000
dog3.jpg	dog	0.8388
dog4.jpg	dog	1.0000
dog5.jpg	dog	1.0000
dog6.jpg	dog	0.9999
dog7.jpg	dog	1.0000
dog8.jpg	dog	1.0000
dog9.jpg	dog	0.9642
dog10.jpg	dog	1.0000

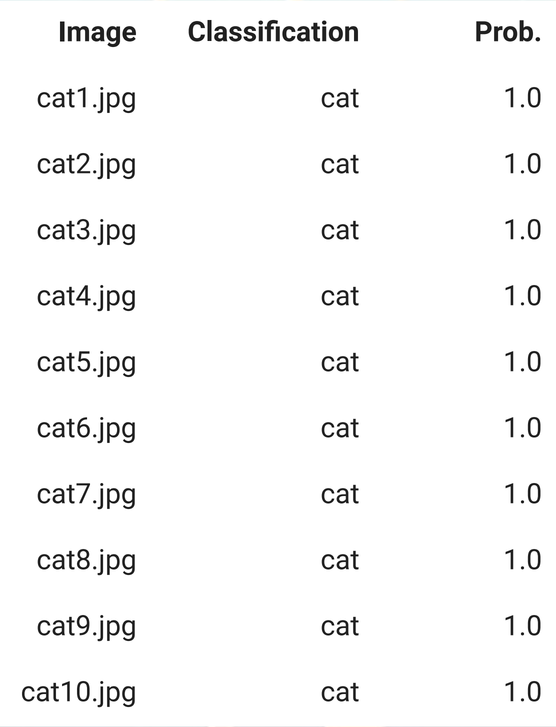


Image	Classification	Prob.
cat1.jpg	cat	1.0
cat2.jpg	cat	1.0
cat3.jpg	cat	1.0
cat4.jpg	cat	1.0
cat5.jpg	cat	1.0
cat6.jpg	cat	1.0
cat7.jpg	cat	1.0
cat8.jpg	cat	1.0
cat9.jpg	cat	1.0
cat10.jpg	cat	1.0

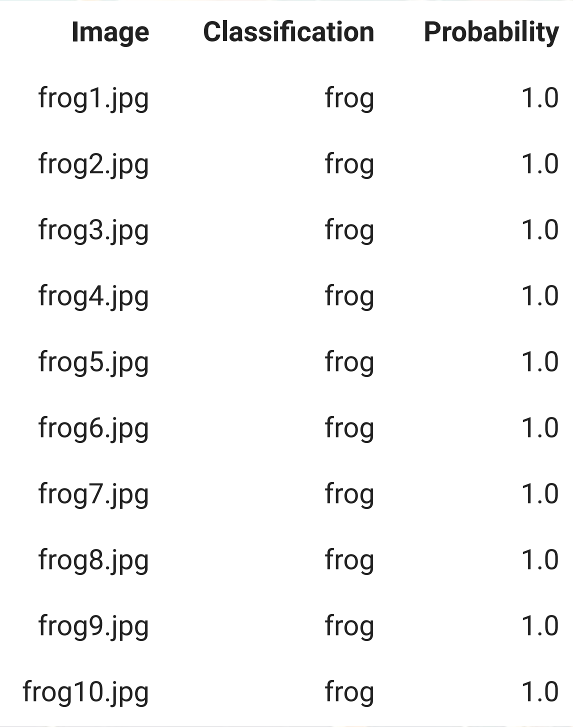


Image	Classification	Probability
frog1.jpg	frog	1.0
frog2.jpg	frog	1.0
frog3.jpg	frog	1.0
frog4.jpg	frog	1.0
frog5.jpg	frog	1.0
frog6.jpg	frog	1.0
frog7.jpg	frog	1.0
frog8.jpg	frog	1.0
frog9.jpg	frog	1.0
frog10.jpg	frog	1.0

Only dogs in test set with models

Densenet121 epoch=10

Image	Classification	Prob
dog1.jpg	dog	1.0000
dog2.jpg	dog	1.0000
dog3.jpg	dog	0.9896
dog4.jpg	dog	1.0000
dog5.jpg	dog	1.0000
dog6.jpg	dog	1.0000
dog7.jpg	dog	1.0000
dog8.jpg	dog	1.0000
dog9.jpg	cat	0.0742
dog10.jpg	dog	1.0000

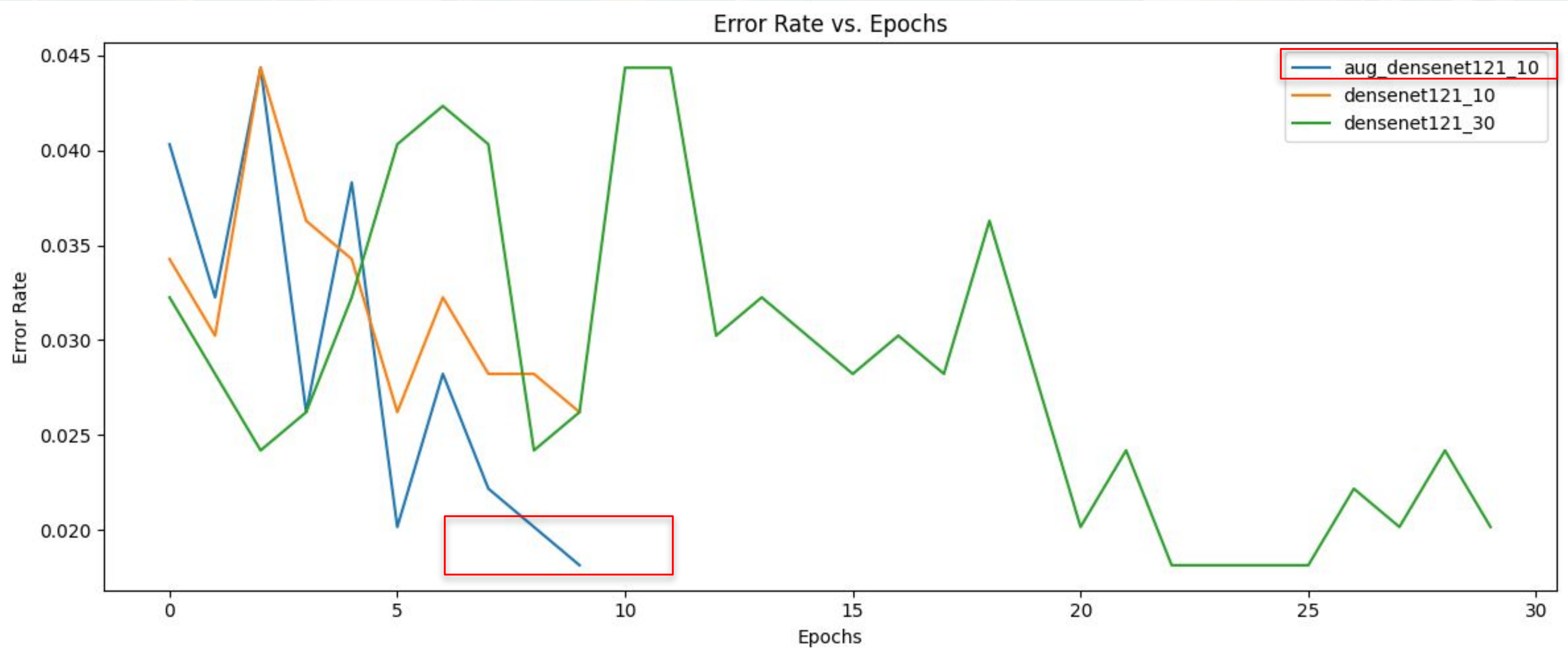
Densenet121 epoch=30

Image	Classification	Prob
dog1.jpg	dog	1.0000
dog2.jpg	dog	1.0000
dog3.jpg	dog	0.7096
dog4.jpg	dog	1.0000
dog5.jpg	dog	1.0000
dog6.jpg	dog	1.0000
dog7.jpg	dog	1.0000
dog8.jpg	dog	1.0000
dog9.jpg	dog	0.9974
dog10.jpg	dog	1.0000

Densenet121 epoch=10 + Augmentation

Image	Classification	Prob
dog1.jpg	dog	1.0000
dog2.jpg	dog	1.0000
dog3.jpg	dog	0.8388
dog4.jpg	dog	1.0000
dog5.jpg	dog	1.0000
dog6.jpg	dog	0.9999
dog7.jpg	dog	1.0000
dog8.jpg	dog	1.0000
dog9.jpg	dog	0.9642
dog10.jpg	dog	1.0000

Error_rate epoch=10/ 30/ 10+augmentations



- Lowest Error_rate is 1.8%, in epoch10 + augmentations

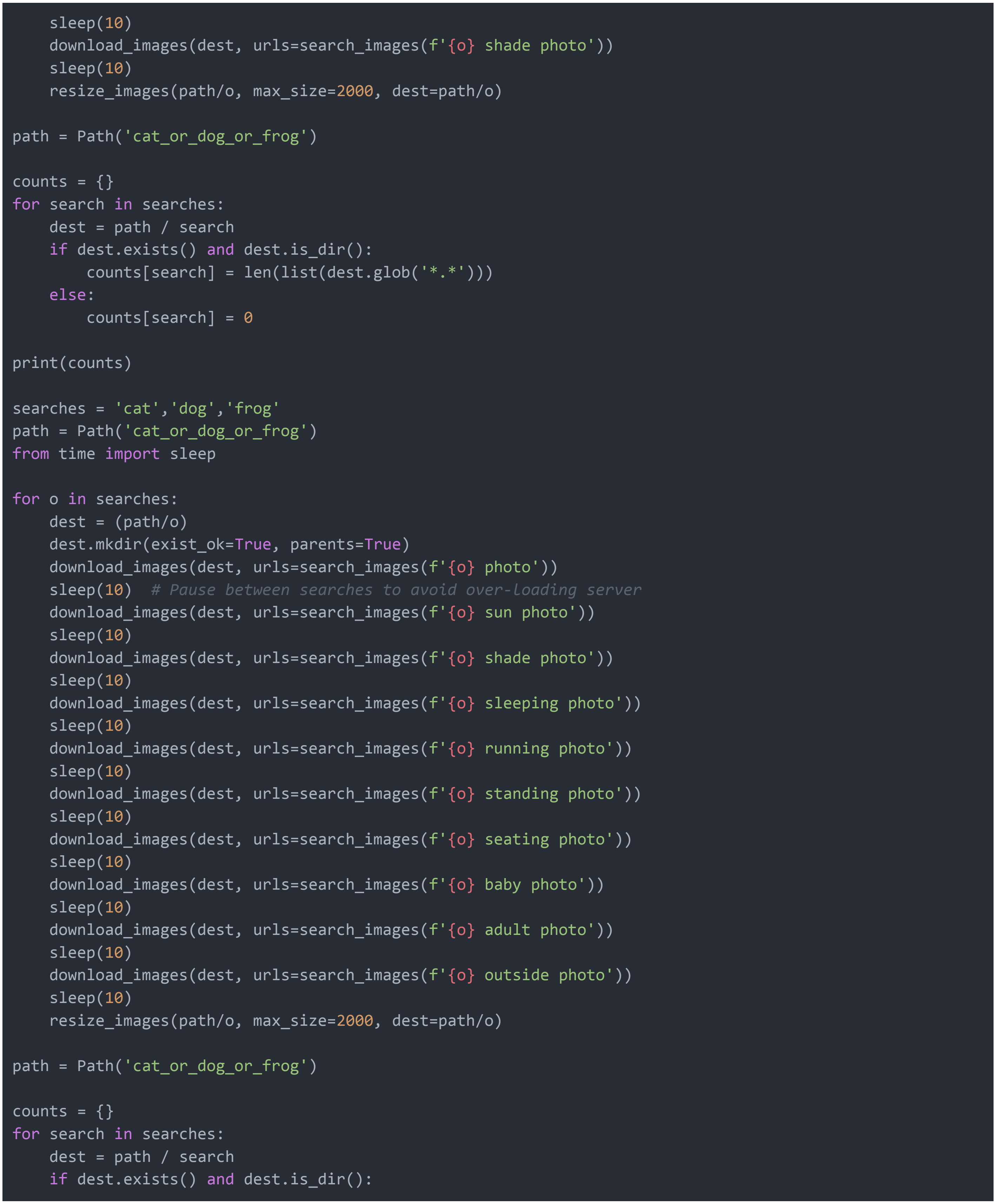
Thanks!

Do you have any questions?

REFERENCE

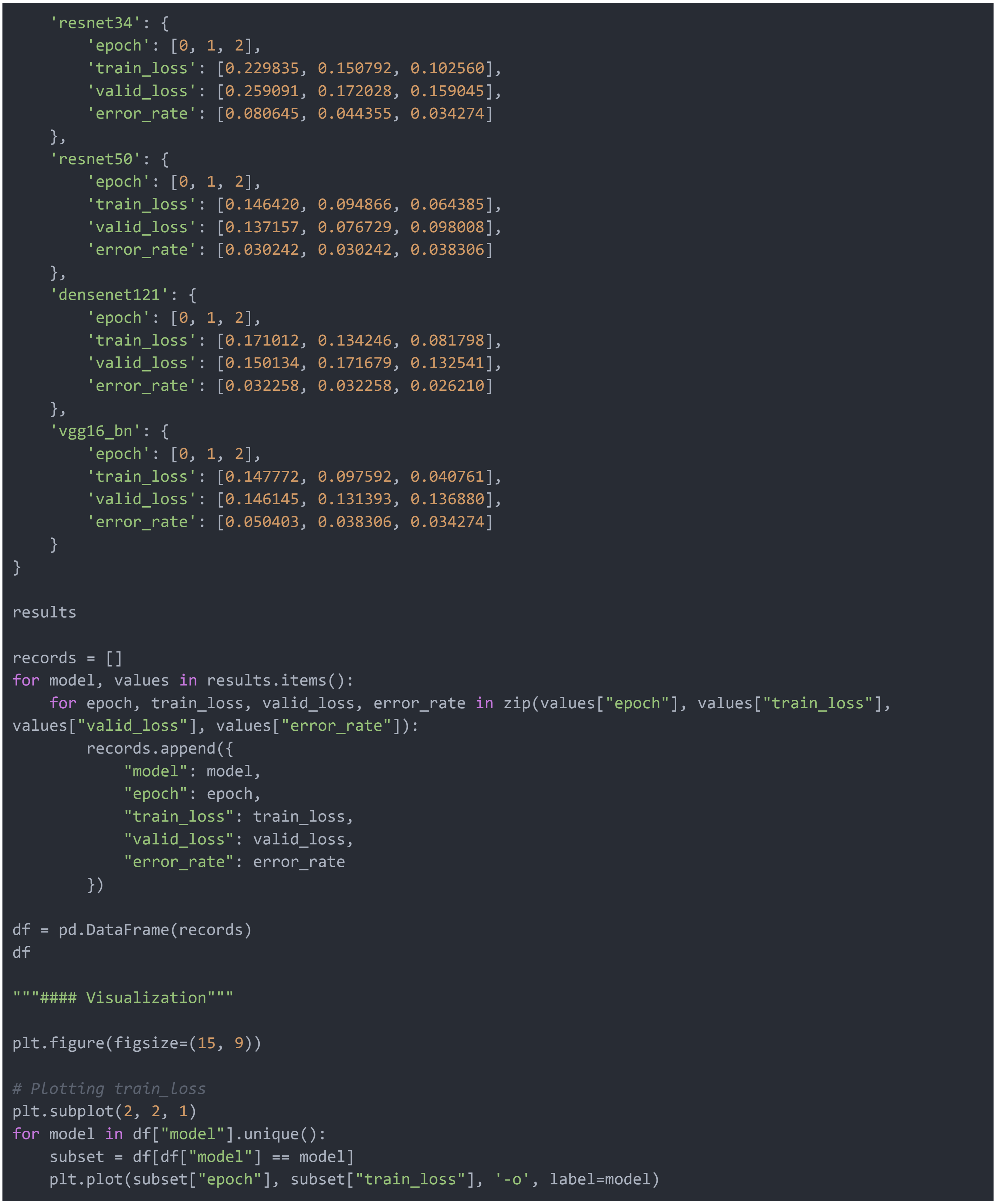
- Howard, J. (n.d.). Is it a bird? Creating a model from your own data. Kaggle. Retrieved from https://www.kaggle.com/code/jhoward/is-it-a-bird-creating-a-model-from-your-own-data#Step-1:-Download-images-of-birds-and-non-birds
- fastai. (n.d.). Documentation. Retrieved from https://docs.fast.ai/
- fastai. (n.d.). Vision. Retrieved from https://fastai1.fast.ai/vision.html
- fastai. (n.d.). Vision learner. Retrieved from https://docs.fast.ai/vision.learner.html
- Madhugiri, V. (2022). Image augmentation using 3 Python libraries. Analytics Vidhya. Retrieved from https://www.analyticsvidhya.com/blog/2022/04/image-augmentation-using-3-python-libraries/
- mathworks. (2023). resnet18. Retrieved from https://www.mathworks.com/help/deeplearning/ref/resnet18.html
- OpenVINO. (2022). Resnet 34 PyTorch. Retrieved from https://docs.openvino.ai/2023.1/omz_models_model_resnet_34_pytorch.html#:~:text=ResNet%2034%20is%20image%20classification,224%2C%20224%20in%20RGB%20order.
- mathworks. (2023). resnet50. Retrieved from https://www.mathworks.com/help/deeplearning/ref/resnet50.html
- Sarkar, T. (2020). Creating DenseNet-121 with TensorFlow. Medium. Retrieved from https://towardsdatascience.com/creating-densenet-121-with-tensorflow-edbc08a956d8#:~:text=DenseNet%20is%20a%20convolutional%20neural,4th%2C%205th%20and%20so%20on.
- Kai, Z. (2011). Error rate. In Springer. Retrieved from https://link.springer.com/referenceworkentry/10.1007/978-0-387-30164-8_262#:~:text=Error%20rate%20refers%20to%20a,respect%20to%20the%20true%20model.

```
# Checking internet or a GPU working
import socket,warnings
try:
    socket.setdefaulttimeout(1)
    socket.socket(socket.AF_INET, socket.SOCK_STREAM).connect(('1.1.1.1', 53))
except socket.error as ex: raise Exception("STOP: No internet. Click '>|' in top right and set 'Internet' switch to on")

"""## IS it a me?"""

pip install duckduckgo_search

from duckduckgo_search import ddg_images
from fastcore.all import *

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from fastai.vision.all import *
from pathlib import Path

def search_images(term, max_images=150):
    print(f"Searching for '{term}'")
    return L(ddg_images(term, max_results=max_images)).itemgot('image')

urls = search_images('cat photo', max_images=1)
urls[0]

from fastdownload import download_url
dest = 'cat.jpg'
download_url(urls[0], dest, show_progress=False)

im = Image.open(dest)
im.to_thumb(256,256)

download_url(search_images('dog photos', max_images=1)[0], 'dog.jpg', show_progress=False)
Image.open('dog.jpg').to_thumb(256,256)

"""#### download the images"""

searches = 'cat','dog','frog'
path = Path('cat_or_dog_or_frog')
from time import sleep

for o in searches:
    dest = (path/o)
    dest.mkdir(exist_ok=True, parents=True)
    download_images(dest, urls=search_images(f'{o} photo'))
    sleep(10)  # Pause between searches to avoid over-loading server
    download_images(dest, urls=search_images(f'{o} sun photo'))
```



```
    sleep(10)
    download_images(dest, urls=search_images(f'{o} shade photo'))
    sleep(10)
    resize_images(path/o, max_size=2000, dest=path/o)

path = Path('cat_or_dog_or_frog')

counts = {}
for search in searches:
    dest = path / search
    if dest.exists() and dest.is_dir():
        counts[search] = len(list(dest.glob('*.*')))
    else:
        counts[search] = 0

print(counts)

searches = 'cat','dog','frog'
path = Path('cat_or_dog_or_frog')
from time import sleep

for o in searches:
    dest = (path/o)
    dest.mkdir(exist_ok=True, parents=True)
    download_images(dest, urls=search_images(f'{o} photo'))
    sleep(10)  # Pause between searches to avoid over-loading server
    download_images(dest, urls=search_images(f'{o} sun photo'))
    sleep(10)
    download_images(dest, urls=search_images(f'{o} shade photo'))
    sleep(10)
    download_images(dest, urls=search_images(f'{o} sleeping photo'))
    sleep(10)
    download_images(dest, urls=search_images(f'{o} running photo'))
    sleep(10)
    download_images(dest, urls=search_images(f'{o} standing photo'))
    sleep(10)
    download_images(dest, urls=search_images(f'{o} seating photo'))
    sleep(10)
    download_images(dest, urls=search_images(f'{o} baby photo'))
    sleep(10)
    download_images(dest, urls=search_images(f'{o} adult photo'))
    sleep(10)
    download_images(dest, urls=search_images(f'{o} outside photo'))
    sleep(10)
    resize_images(path/o, max_size=2000, dest=path/o)

path = Path('cat_or_dog_or_frog')

counts = {}
for search in searches:
    dest = path / search
    if dest.exists() and dest.is_dir():
```

```python
        counts[search] = len(list(dest.glob('*.*')))
    else:
        counts[search] = 0

print(counts)

verify_images()

failed = verify_images(get_image_files(path))
failed.map(Path.unlink)
print(f"{len(failed)} images removed due to verification failure")

verify_images

# Data loading and augmentation
dls = DataBlock(
    blocks=(ImageBlock, CategoryBlock),
    get_items=get_image_files,
    splitter=RandomSplitter(valid_pct=0.2, seed=42),
    get_y=parent_label,
    item_tfms=[Resize(192, method='squish')]
).dataloaders(path, bs=32)

dls.show_batch(max_n=20)

train_images = len(dls.train_ds)

valid_images = len(dls.valid_ds)

total_images = train_images + valid_images

print(f"Number of training images: {train_images}")
print(f"Number of validation images: {valid_images}")
print(f"Total number of images: {total_images}")

models = [resnet18, resnet34, resnet50, densenet121, vgg16_bn]

for model in models:
    print(f"Training using {model.__name__} architecture")

    # Define custom cut values for specific architectures
    if model == densenet121:
        cut = -1  # or another appropriate value
    elif model == vgg16_bn:
        cut = -2  # VGG has two classifier layers at the end
    else:
        cut = None  # let fastai handle it

    learn = vision_learner(dls, model, metrics=error_rate, cut=cut)
    learn.fine_tune(3)

results = {
```



```
    'resnet34': {
        'epoch': [0, 1, 2],
        'train_loss': [0.229835, 0.150792, 0.102560],
        'valid_loss': [0.259091, 0.172028, 0.159045],
        'error_rate': [0.080645, 0.044355, 0.034274]
    },
    'resnet50': {
        'epoch': [0, 1, 2],
        'train_loss': [0.146420, 0.094866, 0.064385],
        'valid_loss': [0.137157, 0.076729, 0.098008],
        'error_rate': [0.030242, 0.030242, 0.038306]
    },
    'densenet121': {
        'epoch': [0, 1, 2],
        'train_loss': [0.171012, 0.134246, 0.081798],
        'valid_loss': [0.150134, 0.171679, 0.132541],
        'error_rate': [0.032258, 0.032258, 0.026210]
    },
    'vgg16_bn': {
        'epoch': [0, 1, 2],
        'train_loss': [0.147772, 0.097592, 0.040761],
        'valid_loss': [0.146145, 0.131393, 0.136880],
        'error_rate': [0.050403, 0.038306, 0.034274]
    }
}

results

records = []
for model, values in results.items():
    for epoch, train_loss, valid_loss, error_rate in zip(values["epoch"], values["train_loss"],
values["valid_loss"], values["error_rate"]):
        records.append({
            "model": model,
            "epoch": epoch,
            "train_loss": train_loss,
            "valid_loss": valid_loss,
            "error_rate": error_rate
        })

df = pd.DataFrame(records)
df

"""#### Visualization"""

plt.figure(figsize=(15, 9))

# Plotting train_loss
plt.subplot(2, 2, 1)
for model in df["model"].unique():
    subset = df[df["model"] == model]
    plt.plot(subset["epoch"], subset["train_loss"], '-o', label=model)
plt.title("Train Loss vs. Epoch")
plt.xlabel("Epoch")
plt.ylabel("Train Loss")
plt.legend()
plt.grid(True)

# Plotting valid_loss
plt.subplot(2, 2, 2)
for model in df["model"].unique():
    subset = df[df["model"] == model]
    plt.plot(subset["epoch"], subset["valid_loss"], '-o', label=model)
plt.title("Valid Loss vs. Epoch")
plt.xlabel("Epoch")
plt.ylabel("Valid Loss")
plt.legend()
plt.grid(True)

# Plotting error_rate
plt.subplot(2, 2, 3)
for model in df["model"].unique():
    subset = df[df["model"] == model]
    plt.plot(subset["epoch"], subset["error_rate"], '-o', label=model)
plt.title("Error Rate vs. Epoch")
plt.xlabel("Epoch")
plt.ylabel("Error Rate")
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()

"""#### Testing images in google different from training images"""

from fastai.metrics import error_rate

"""#### adding epoch to 10"""

learn = vision_learner(dls, densenet121, metrics=error_rate)
learn.fine_tune(10)

results_densenet121_10 = {
        'epoch': [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
        'train_loss': [0.099445, 0.076969, 0.056807, 0.077282, 0.068521, 0.034743, 0.017370, 0.008646,
0.012727, 0.010398],
        'valid_loss': [0.112311, 0.175730, 0.202622, 0.138919, 0.197403, 0.170971, 0.128582, 0.150120,
0.155263, 0.152180],
        'error_rate': [0.034274, 0.030242, 0.044355, 0.036290, 0.034274, 0.026210, 0.032258, 0.028226,
0.028226, 0.026210]
    }

df_densenet121_10 = pd.DataFrame({
    "model": ["densenet121"] * 10,
    "epoch": results_densenet121_10['epoch'],
    "train_loss": results_densenet121_10["train_loss"],
    "valid_loss": results_densenet121_10["valid_loss"],
    "error_rate": results_densenet121_10["error_rate"]
})

plt.figure(figsize=(20, 6))

# Plotting train_loss
plt.subplot(1, 3, 1)
plt.plot(df_densenet121_10["epoch"], df_densenet121_10["train_loss"], '-o', label="Train Loss")
plt.title("Train Loss vs. Epoch")
plt.xlabel("Epoch")
plt.ylabel("Train Loss")
plt.legend()
plt.grid(True)

# Plotting valid_loss
plt.subplot(1, 3, 2)
plt.plot(df_densenet121_10["epoch"], df_densenet121_10["valid_loss"], '-o', label="Validation Loss",
color="orange")
plt.title("Validation Loss vs. Epoch")
plt.xlabel("Epoch")
plt.ylabel("Validation Loss")
plt.legend()
plt.grid(True)

# Plotting error_rate
plt.subplot(1, 3, 3)
plt.plot(df_densenet121_10["epoch"], df_densenet121_10["error_rate"], '-o', label="Error Rate",
color="green")
plt.title("Error Rate vs. Epoch")
plt.xlabel("Epoch")
plt.ylabel("Error Rate")
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()

results_densenet121_10 = {
    'epoch': [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'train_loss': [1.045902, 1.001012, 0.925600, 0.756249, 0.606398, 0.504028, 0.417575, 0.354688,
0.307628, 0.272610, 0.243759],
    'error_rate': [0.416667, 0.500000, 0.416667, 0.333333, 0.250000, 0.250000, 0.250000, 0.166667,
0.166667, 0.166667, 0.166667]
}

"""#### check the misprediction"""

from fastai.interpret import ClassificationInterpretation
```

```
interp = ClassificationInterpretation.from_learner(learn)

interp.plot_top_losses(k=20, figsize=(15,10))

"""#### Test model with other images"""

from google.colab import drive
drive.mount('/content/drive')

test_images = [f'dog{i}.jpg' for i in range(1, 11)]

for img in test_images:
    img_path = f'/content/drive/My Drive/test/{img}'
    dog, _, probs = learn.predict(PILImage.create(img_path))
    print(f"{img} is classified as: {dog}. Probability it's dog: {probs[1]:.4f}")
    print("-"*50)

import re

text = """
dog1.jpg is classified as: dog. Probability it's dog: 1.0000
--------------------------------------------------
dog2.jpg is classified as: dog. Probability it's dog: 1.0000
--------------------------------------------------
dog3.jpg is classified as: dog. Probability it's dog: 0.9896
--------------------------------------------------
dog4.jpg is classified as: dog. Probability it's dog: 1.0000
--------------------------------------------------
dog5.jpg is classified as: dog. Probability it's dog: 1.0000
--------------------------------------------------
dog6.jpg is classified as: dog. Probability it's dog: 1.0000
--------------------------------------------------
dog7.jpg is classified as: dog. Probability it's dog: 1.0000
--------------------------------------------------
dog8.jpg is classified as: dog. Probability it's dog: 1.0000
--------------------------------------------------
dog9.jpg is classified as: cat. Probability it's dog: 0.0742
--------------------------------------------------
dog10.jpg is classified as: dog. Probability it's dog: 1.0000
--------------------------------------------------
"""

image_names = re.findall(r"(dog\d+\.jpg)", text)
classifications = re.findall(r"classified as: (\w+)", text)
probabilities = [float(p) for p in re.findall(r"Probability it's dog: (\d+\.\d+)", text)]

data1 = {
    'Image': image_names,
    'Classification': classifications,
    'Probability': probabilities
}
```

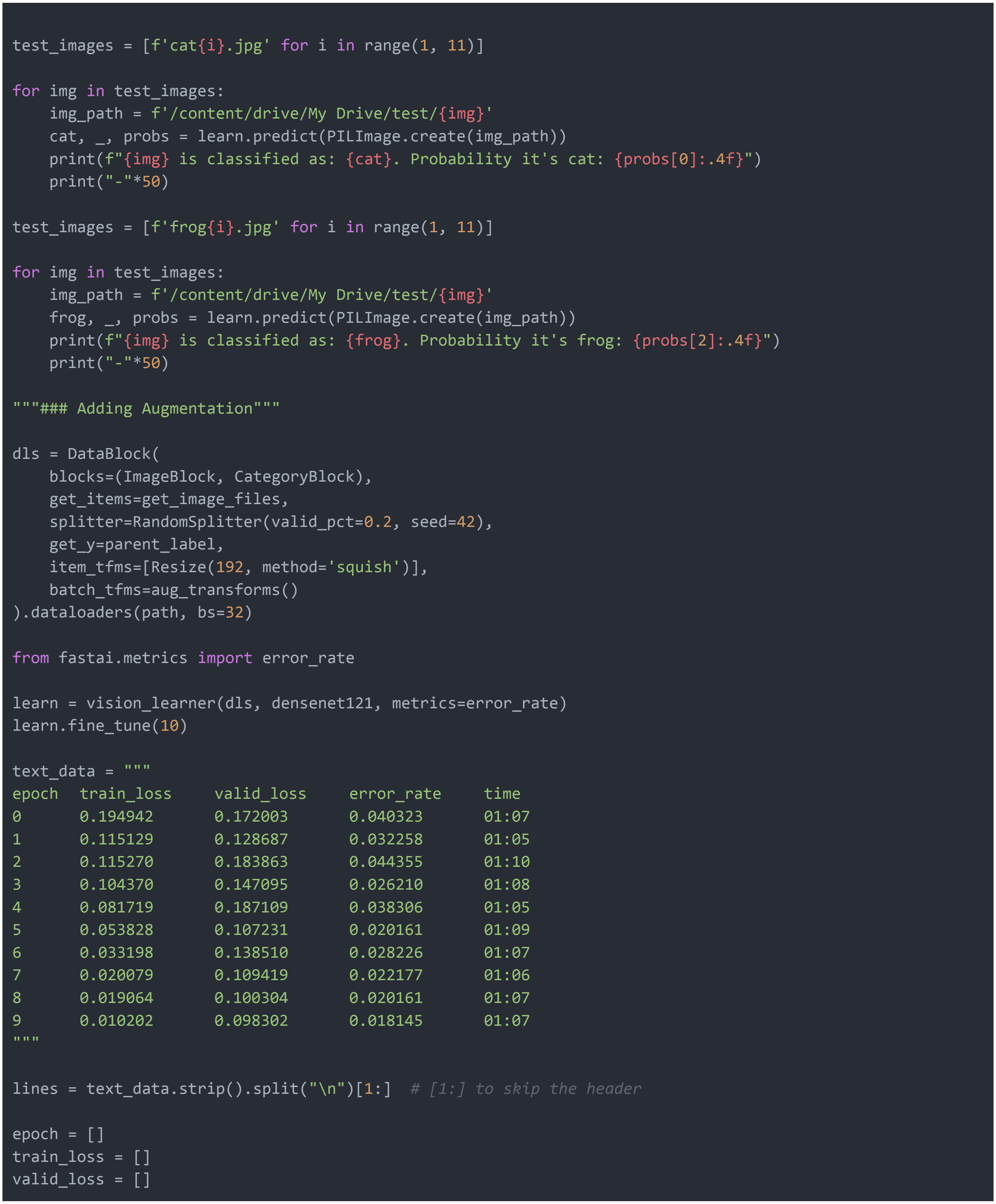
```python
data1

df_classification1 = pd.DataFrame(data1)
df_classification1

"""#### To cat"""

test_images = [f'cat{i}.jpg' for i in range(1, 11)]

for img in test_images:
    img_path = f'/content/drive/My Drive/test/{img}'
    cat, _, probs = learn.predict(PILImage.create(img_path))
    print(f"{img} is classified as: {cat}. Probability it's cat: {probs[0]:.4f}")
    print("-"*50)

text = """
cat1.jpg is classified as: cat. Probability it's cat: 1.0000
--------------------------------------------------
cat2.jpg is classified as: cat. Probability it's cat: 1.0000
--------------------------------------------------
cat3.jpg is classified as: cat. Probability it's cat: 1.0000
--------------------------------------------------
cat4.jpg is classified as: cat. Probability it's cat: 1.0000
--------------------------------------------------
cat5.jpg is classified as: cat. Probability it's cat: 1.0000
--------------------------------------------------
cat6.jpg is classified as: cat. Probability it's cat: 1.0000
--------------------------------------------------
cat7.jpg is classified as: cat. Probability it's cat: 1.0000
--------------------------------------------------
cat8.jpg is classified as: cat. Probability it's cat: 1.0000
--------------------------------------------------
cat9.jpg is classified as: cat. Probability it's cat: 1.0000
--------------------------------------------------
cat10.jpg is classified as: cat. Probability it's cat: 1.0000
--------------------------------------------------
"""

image_names = re.findall(r"(cat\d+\.jpg)", text)
classifications = re.findall(r"classified as: (\w+)", text)
probabilities = [float(p) for p in re.findall(r"Probability it's cat: (\d+\.\d+)", text)]

data2 = {
    'Image': image_names,
    'Classification': classifications,
    'Probability': probabilities
}

data2

df_classification2 = pd.DataFrame(data2)
df_classification2
```

```
"""#### to frog"""

test_images = [f'frog{i}.jpg' for i in range(1, 11)]

for img in test_images:
    img_path = f'/content/drive/My Drive/test/{img}'
    frog, _, probs = learn.predict(PILImage.create(img_path))
    print(f"{img} is classified as: {frog}. Probability it's frog: {probs[2]:.4f}")
    print("-"*50)

text = """
frog1.jpg is classified as: frog. Probability it's frog: 1.0000
--------------------------------------------------
frog2.jpg is classified as: frog. Probability it's frog: 1.0000
--------------------------------------------------
frog3.jpg is classified as: frog. Probability it's frog: 1.0000
--------------------------------------------------
frog4.jpg is classified as: frog. Probability it's frog: 1.0000
--------------------------------------------------
frog5.jpg is classified as: frog. Probability it's frog: 1.0000
--------------------------------------------------
frog6.jpg is classified as: frog. Probability it's frog: 1.0000
--------------------------------------------------
frog7.jpg is classified as: frog. Probability it's frog: 1.0000
--------------------------------------------------
frog8.jpg is classified as: frog. Probability it's frog: 1.0000
--------------------------------------------------
frog9.jpg is classified as: frog. Probability it's frog: 1.0000
--------------------------------------------------
frog10.jpg is classified as: frog. Probability it's frog: 1.0000
--------------------------------------------------
"""

image_names = re.findall(r"(frog\d+\.jpg)", text)
classifications = re.findall(r"classified as: (\w+)", text)
probabilities = [float(p) for p in re.findall(r"Probability it's frog: (\d+\.\d+)", text)]

data3 = {
    'Image': image_names,
    'Classification': classifications,
    'Probability': probabilities
}

data3

df_classification3 = pd.DataFrame(data3)
df_classification3

"""#### adding epoch to 30"""

from fastai.metrics import error_rate
```

```
learn = vision_learner(dls, densenet121, metrics=error_rate)
learn.fine_tune(30)

text_data = """
epoch   train_loss      valid_loss      error_rate      time
0       0.095793        0.108928        0.032258        01:16
1       0.051865        0.092819        0.028226        01:16
2       0.034021        0.099827        0.024194        01:03
3       0.034106        0.114688        0.026210        01:06
4       0.025709        0.149411        0.032258        01:04
5       0.043007        0.197642        0.040323        01:07
6       0.054408        0.186182        0.042339        01:06
7       0.048845        0.188962        0.040323        01:04
8       0.024546        0.145037        0.024194        01:07
9       0.036698        0.141662        0.026210        01:04
10      0.036303        0.227877        0.044355        01:13
11      0.039621        0.241808        0.044355        01:20
12      0.027854        0.163183        0.030242        01:27
13      0.015833        0.174244        0.032258        01:15
14      0.018007        0.177479        0.030242        01:09
15      0.008182        0.170533        0.028226        01:10
16      0.005399        0.172935        0.030242        01:06
17      0.006652        0.152392        0.028226        01:06
18      0.011072        0.222670        0.036290        01:03
19      0.005197        0.185639        0.028226        01:06
20      0.004634        0.164897        0.020161        01:08
21      0.003358        0.183613        0.024194        01:04
22      0.002495        0.156465        0.018145        01:07
23      0.001241        0.165297        0.018145        01:08
24      0.001079        0.144529        0.018145        01:04
25      0.001184        0.147033        0.018145        01:06
26      0.000660        0.152218        0.022177        01:04
27      0.005207        0.149952        0.020161        01:08
28      0.001900        0.160705        0.024194        01:06
29      0.003383        0.153165        0.020161        01:06
"""

lines = text_data.strip().split("\n")[1:]  # [1:] to skip the header

epoch = []
train_loss = []
valid_loss = []
error_rate = []

for line in lines:
    parts = line.split()
    epoch.append(int(parts[0]))
    train_loss.append(float(parts[1]))
    valid_loss.append(float(parts[2]))
    error_rate.append(float(parts[3]))
```

```
results_densenet121_30 = {
    'epoch': epoch,
    'train_loss': train_loss,
    'valid_loss': valid_loss,
    'error_rate': error_rate
}

df_densenet121_30 = pd.DataFrame({
    "model": ["densenet121"] * 30,
    "epoch": results_densenet121_30['epoch'],
    "train_loss": results_densenet121_30["train_loss"],
    "valid_loss": results_densenet121_30["valid_loss"],
    "error_rate": results_densenet121_30["error_rate"]
})

plt.figure(figsize=(20, 6))

# Plotting train_loss
plt.subplot(1, 3, 1)
plt.plot(df_densenet121_30["epoch"], df_densenet121_30["train_loss"], '-o', label="Train Loss")
plt.title("Train Loss vs. Epoch")
plt.xlabel("Epoch")
plt.ylabel("Train Loss")
plt.legend()
plt.grid(True)

# Plotting valid_loss
plt.subplot(1, 3, 2)
plt.plot(df_densenet121_30["epoch"], df_densenet121_30["valid_loss"], '-o', label="Validation Loss",
color="orange")
plt.title("Validation Loss vs. Epoch")
plt.xlabel("Epoch")
plt.ylabel("Validation Loss")
plt.legend()
plt.grid(True)

# Plotting error_rate
plt.subplot(1, 3, 3)
plt.plot(df_densenet121_30["epoch"], df_densenet121_30["error_rate"], '-o', label="Error Rate",
color="green")
plt.title("Error Rate vs. Epoch")
plt.xlabel("Epoch")
plt.ylabel("Error Rate")
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()

"""#### Check misprediction"""

interp = ClassificationInterpretation.from_learner(learn)
```

```python
interp.plot_top_losses(k=20, figsize=(15,10))

"""#### Checking dog"""

test_images = [f'dog{i}.jpg' for i in range(1, 11)]

for img in test_images:
    img_path = f'/content/drive/My Drive/test/{img}'
    dog, _, probs = learn.predict(PILImage.create(img_path))
    print(f"{img} is classified as: {dog}. Probability it's dog: {probs[1]:.4f}")
    print("-"*50)

text = """
dog1.jpg is classified as: dog. Probability it's dog: 1.0000
--------------------------------------------------
dog2.jpg is classified as: dog. Probability it's dog: 1.0000
--------------------------------------------------
dog3.jpg is classified as: dog. Probability it's dog: 0.7096
--------------------------------------------------
dog4.jpg is classified as: dog. Probability it's dog: 1.0000
--------------------------------------------------
dog5.jpg is classified as: dog. Probability it's dog: 1.0000
--------------------------------------------------
dog6.jpg is classified as: dog. Probability it's dog: 1.0000
--------------------------------------------------
dog7.jpg is classified as: dog. Probability it's dog: 1.0000
--------------------------------------------------
dog8.jpg is classified as: dog. Probability it's dog: 1.0000
--------------------------------------------------
dog9.jpg is classified as: dog. Probability it's dog: 0.9974
--------------------------------------------------
dog10.jpg is classified as: dog. Probability it's dog: 1.0000
--------------------------------------------------
"""

image_names = re.findall(r"(dog\d+\.jpg)", text)
classifications = re.findall(r"classified as: (\w+)", text)
probabilities = [float(p) for p in re.findall(r"Probability it's dog: (\d+\.\d+)", text)]

data4 = {
    'Image': image_names,
    'Classification': classifications,
    'Probability': probabilities
}

data4

df_classification4 = pd.DataFrame(data4)
df_classification4

"""#### Others"""
```



```
test_images = [f'cat{i}.jpg' for i in range(1, 11)]

for img in test_images:
    img_path = f'/content/drive/My Drive/test/{img}'
    cat, _, probs = learn.predict(PILImage.create(img_path))
    print(f"{img} is classified as: {cat}. Probability it's cat: {probs[0]:.4f}")
    print("-"*50)

test_images = [f'frog{i}.jpg' for i in range(1, 11)]

for img in test_images:
    img_path = f'/content/drive/My Drive/test/{img}'
    frog, _, probs = learn.predict(PILImage.create(img_path))
    print(f"{img} is classified as: {frog}. Probability it's frog: {probs[2]:.4f}")
    print("-"*50)

"""### Adding Augmentation"""

dls = DataBlock(
    blocks=(ImageBlock, CategoryBlock),
    get_items=get_image_files,
    splitter=RandomSplitter(valid_pct=0.2, seed=42),
    get_y=parent_label,
    item_tfms=[Resize(192, method='squish')],
    batch_tfms=aug_transforms()
).dataloaders(path, bs=32)

from fastai.metrics import error_rate

learn = vision_learner(dls, densenet121, metrics=error_rate)
learn.fine_tune(10)

text_data = """
epoch   train_loss     valid_loss     error_rate     time
0       0.194942       0.172003       0.040323       01:07
1       0.115129       0.128687       0.032258       01:05
2       0.115270       0.183863       0.044355       01:10
3       0.104370       0.147095       0.026210       01:08
4       0.081719       0.187109       0.038306       01:05
5       0.053828       0.107231       0.020161       01:09
6       0.033198       0.138510       0.028226       01:07
7       0.020079       0.109419       0.022177       01:06
8       0.019064       0.100304       0.020161       01:07
9       0.010202       0.098302       0.018145       01:07
"""

lines = text_data.strip().split("\n")[1:]  # [1:] to skip the header

epoch = []
train_loss = []
valid_loss = []
```

```python
error_rate = []

for line in lines:
    parts = line.split()
    epoch.append(int(parts[0]))
    train_loss.append(float(parts[1]))
    valid_loss.append(float(parts[2]))
    error_rate.append(float(parts[3]))

aug_densenet121_10 = {
    'epoch': epoch,
    'train_loss': train_loss,
    'valid_loss': valid_loss,
    'error_rate': error_rate
}

df_aug_densenet121_10 = pd.DataFrame({
    "model": ["densenet121"] * 10,
    "epoch": aug_densenet121_10['epoch'],
    "train_loss": aug_densenet121_10["train_loss"],
    "valid_loss": aug_densenet121_10["valid_loss"],
    "error_rate": aug_densenet121_10["error_rate"]
})

plt.figure(figsize=(20, 6))

# Plotting train_loss
plt.subplot(1, 3, 1)
plt.plot(df_aug_densenet121_10["epoch"], df_aug_densenet121_10["train_loss"], '-o', label="Train Loss")
plt.title("Train Loss vs. Epoch")
plt.xlabel("Epoch")
plt.ylabel("Train Loss")
plt.legend()
plt.grid(True)

# Plotting valid_loss
plt.subplot(1, 3, 2)
plt.plot(df_aug_densenet121_10["epoch"], df_aug_densenet121_10["valid_loss"], '-o', label="Validation Loss", color="orange")
plt.title("Validation Loss vs. Epoch")
plt.xlabel("Epoch")
plt.ylabel("Validation Loss")
plt.legend()
plt.grid(True)

# Plotting error_rate
plt.subplot(1, 3, 3)
plt.plot(df_aug_densenet121_10["epoch"], df_aug_densenet121_10["error_rate"], '-o', label="Error Rate", color="green")
plt.title("Error Rate vs. Epoch")
plt.xlabel("Epoch")
plt.ylabel("Error Rate")
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()

"""#### Check misprediction"""

interp = ClassificationInterpretation.from_learner(learn)

interp.plot_top_losses(k=20, figsize=(15,10))

"""#### Checking dog"""

test_images = [f'dog{i}.jpg' for i in range(1, 11)]

for img in test_images:
    img_path = f'/content/drive/My Drive/test/{img}'
    dog, _, probs = learn.predict(PILImage.create(img_path))
    print(f"{img} is classified as: {dog}. Probability it's dog: {probs[1]:.4f}")
    print("-"*50)

text = """
dog1.jpg is classified as: dog. Probability it's dog: 1.0000
--------------------------------------------------
dog2.jpg is classified as: dog. Probability it's dog: 1.0000
--------------------------------------------------
dog3.jpg is classified as: dog. Probability it's dog: 0.8388
--------------------------------------------------
dog4.jpg is classified as: dog. Probability it's dog: 1.0000
--------------------------------------------------
dog5.jpg is classified as: dog. Probability it's dog: 1.0000
--------------------------------------------------
dog6.jpg is classified as: dog. Probability it's dog: 0.9999
--------------------------------------------------
dog7.jpg is classified as: dog. Probability it's dog: 1.0000
--------------------------------------------------
dog8.jpg is classified as: dog. Probability it's dog: 1.0000
--------------------------------------------------
dog9.jpg is classified as: dog. Probability it's dog: 0.9642
--------------------------------------------------
dog10.jpg is classified as: dog. Probability it's dog: 1.0000
--------------------------------------------------
"""

image_names = re.findall(r"(dog\d+\.jpg)", text)
classifications = re.findall(r"classified as: (\w+)", text)
probabilities = [float(p) for p in re.findall(r"Probability it's dog: (\d+\.\d+)", text)]

data5 = {
    'Image': image_names,
    'Classification': classifications,
    'Probability': probabilities
}

data5

df_classification5 = pd.DataFrame(data5)
df_classification5



"""#### Others"""

test_images = [f'cat{i}.jpg' for i in range(1, 11)]

for img in test_images:
    img_path = f'/content/drive/My Drive/test/{img}'
    cat, _, probs = learn.predict(PILImage.create(img_path))
    print(f"{img} is classified as: {cat}. Probability it's cat: {probs[0]:.4f}")
    print("-"*50)

test_images = [f'frog{i}.jpg' for i in range(1, 11)]

for img in test_images:
    img_path = f'/content/drive/My Drive/test/{img}'
    frog, _, probs = learn.predict(PILImage.create(img_path))
    print(f"{img} is classified as: {frog}. Probability it's frog: {probs[2]:.4f}")
    print("-"*50)

"""#### Final Visualization"""

df_densenet121_10 = pd.DataFrame({
    "model": ["densenet121_10"] * 10,
    "epoch": results_densenet121_10['epoch'],
    "train_loss": results_densenet121_10["train_loss"],
    "valid_loss": results_densenet121_10["valid_loss"],
    "error_rate": results_densenet121_10["error_rate"]
})

df_densenet121_30 = pd.DataFrame({
    "model": ["densenet121_30"] * 30,
    "epoch": results_densenet121_30['epoch'],
    "train_loss": results_densenet121_30["train_loss"],
    "valid_loss": results_densenet121_30["valid_loss"],
    "error_rate": results_densenet121_30["error_rate"]
})

df_aug_densenet121_10 = pd.DataFrame({
    "model": ["aug_densenet121_10"] * 10,
    "epoch": aug_densenet121_10['epoch'],
    "train_loss": aug_densenet121_10["train_loss"],
    "valid_loss": aug_densenet121_10["valid_loss"],
    "error_rate": aug_densenet121_10["error_rate"]
})

df_all = pd.concat([df_densenet121_10, df_densenet121_30, df_aug_densenet121_10])

fig, ax = plt.subplots(3, 1, figsize=(12, 15))

for model, group in df_all.groupby("model"):
    ax[0].plot(group["epoch"], group["train_loss"], label=model)
    ax[1].plot(group["epoch"], group["valid_loss"], label=model)
    ax[2].plot(group["epoch"], group["error_rate"], label=model)

ax[0].set_title("Train Loss vs. Epochs")
ax[0].set_xlabel("Epochs")
ax[0].set_ylabel("Train Loss")
ax[0].legend()

ax[1].set_title("Validation Loss vs. Epochs")
ax[1].set_xlabel("Epochs")
ax[1].set_ylabel("Validation Loss")
ax[1].legend()

ax[2].set_title("Error Rate vs. Epochs")
ax[2].set_xlabel("Epochs")
ax[2].set_ylabel("Error Rate")
ax[2].legend()

plt.tight_layout()
plt.show()

"""### Epoch 50 and Augmentation"""

dls = DataBlock(
    blocks=(ImageBlock, CategoryBlock),
    get_items=get_image_files,
    splitter=RandomSplitter(valid_pct=0.2, seed=42),
    get_y=parent_label,
    item_tfms=[Resize(192, method='squish')],
    batch_tfms=aug_transforms()
).dataloaders(path, bs=32)

from fastai.metrics import error_rate

learn = vision_learner(dls, densenet121, metrics=error_rate)
learn.fine_tune(50)

text_data = """
epoch   train_loss     valid_loss     error_rate     time
0       0.145881       0.139931       0.040323       01:06
1       0.126715       0.098878       0.030242       01:04
2       0.077759       0.101689       0.028226       01:06
3       0.064724       0.121918       0.026210       01:04
4       0.066769       0.125793       0.022177       01:11
```

```
5      0.054425       0.099485       0.020161       01:07
6      0.035887       0.198499       0.034274       01:04
7      0.037642       0.206767       0.034274       01:06
8      0.058760       0.167597       0.030242       01:08
9      0.049893       0.205076       0.032258       01:04
10     0.049846       0.159486       0.030242       01:06
11     0.035203       0.178444       0.038306       01:04
12     0.032576       0.216624       0.038306       01:06
13     0.062905       0.131325       0.030242       01:06
14     0.040325       0.157208       0.036290       01:05
15     0.043859       0.162575       0.032258       01:06
16     0.049489       0.167094       0.038306       01:04
17     0.030197       0.134751       0.038306       01:06
18     0.030922       0.131850       0.036290       01:04
19     0.021449       0.220281       0.038306       01:07
20     0.023944       0.138701       0.028226       01:10
21     0.034987       0.159316       0.028226       01:04
22     0.015638       0.162622       0.024194       01:07
23     0.022401       0.180803       0.034274       01:07
24     0.032185       0.214561       0.032258       01:07
25     0.018168       0.149021       0.026210       01:07
26     0.018048       0.205449       0.034274       01:07
27     0.009471       0.191617       0.030242       01:05
28     0.009474       0.165191       0.026210       01:09
29     0.005037       0.182198       0.028226       01:06
30     0.015277       0.149766       0.024194       01:06
31     0.011761       0.178169       0.020161       01:08
32     0.009073       0.170063       0.026210       01:07
33     0.007282       0.227717       0.030242       01:07
34     0.008994       0.207989       0.030242       01:07
35     0.007227       0.218286       0.034274       01:05
36     0.004540       0.201531       0.028226       01:08
37     0.002050       0.204395       0.034274       01:07
38     0.001808       0.180580       0.032258       01:06
39     0.001622       0.193618       0.032258       01:08
40     0.001868       0.201432       0.034274       01:12
41     0.002845       0.209718       0.038306       01:07
42     0.001354       0.201751       0.036290       01:07
43     0.001227       0.198977       0.034274       01:10
44     0.000692       0.185016       0.036290       01:10
45     0.000765       0.187105       0.032258       01:08
46     0.000384       0.189292       0.028226       01:06
47     0.001247       0.180472       0.032258       01:10
48     0.001070       0.183826       0.032258       01:08
49     0.001236       0.185454       0.036290       01:05
"""

lines = text_data.strip().split("\n")[1:]  # [1:] to skip the header
epoch = []
train_loss = []
valid_loss = []
error_rate = []
```

```
for line in lines:
    parts = line.split()
    epoch.append(int(parts[0]))
    train_loss.append(float(parts[1]))
    valid_loss.append(float(parts[2]))
    error_rate.append(float(parts[3]))

aug_densenet121_50 = {
    'epoch': epoch,
    'train_loss': train_loss,
    'valid_loss': valid_loss,
    'error_rate': error_rate
}

df_aug_densenet121_50 = pd.DataFrame({
    "model": ["densenet121"] * 50,
    "epoch": aug_densenet121_50['epoch'],
    "train_loss": aug_densenet121_50["train_loss"],
    "valid_loss": aug_densenet121_50["valid_loss"],
    "error_rate": aug_densenet121_50["error_rate"]
})

plt.figure(figsize=(20, 6))

# Plotting train_loss
plt.subplot(1, 3, 1)
plt.plot(df_aug_densenet121_50["epoch"], df_aug_densenet121_50["train_loss"], '-o', label="Train Loss")
plt.title("Train Loss vs. Epoch")
plt.xlabel("Epoch")
plt.ylabel("Train Loss")
plt.legend()
plt.grid(True)

# Plotting valid_loss
plt.subplot(1, 3, 2)
plt.plot(df_aug_densenet121_50["epoch"], df_aug_densenet121_50["valid_loss"], '-o', label="Validation
Loss", color="orange")
plt.title("Validation Loss vs. Epoch")
plt.xlabel("Epoch")
plt.ylabel("Validation Loss")
plt.legend()
plt.grid(True)

# Plotting error_rate
plt.subplot(1, 3, 3)
plt.plot(df_aug_densenet121_50["epoch"], df_aug_densenet121_50["error_rate"], '-o', label="Error Rate",
color="green")
plt.title("Error Rate vs. Epoch")
plt.xlabel("Epoch")
plt.ylabel("Error Rate")
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()

"""#### Check misprediction"""

interp = ClassificationInterpretation.from_learner(learn)

interp.plot_top_losses(k=20, figsize=(15,10))
```