In [2]:
```
# Determine active environment
import sys
print(sys.executable)
```

```
C:\Users\mathy\anaconda3\envs\py311env\python.exe
```

In [ ]:
```
# CIFAR-10 dataset contains 60,000 color images of 32 x 32 px
# 3 channels into 10 classes (outputs)
# 50,000 for training & 10,000 for testing
# https://www.cs.toronto.edu/~kriz/cifar.html
# CDNN: Convoluted Deep Neural Network (with data augmentation)
""" NOTE: 'preview' directory added to root """

"""
# Keras imports (replaced by tensorflow)
from keras.datasets import cifar10
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.optimizers import SGD, Adam, RMSprop
"""

from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense, Dropout, Activation, Conv2D, MaxPooling2D
from tensorflow.keras.optimizers import SGD, Adam, RMSprop
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np
import matplotlib.pyplot as plt
#============================================================================================
# hyperparameters
NUM_TO_AUGMENT = 5
IMG_CHANNELS = 3
IMG_ROWS = 32
IMG_COLS = 32
BATCH_SIZE = 128
N_EPOCHS = 50
N_CLASSES = 10
VERBOSE = 1
VALIDATION_SPLIT = 0.2
OPTIM = RMSprop()

# load dataset
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
print('X_train shape:', X_train.shape)
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')
#============================================================================================
# data augmentation
# TODO: add progress bar for augmentation status
print("Augmenting training set images...")

datagen = ImageDataGenerator(
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
```

```
    fill_mode='nearest'
)

xtas, ytas = [], []

for i in range(X_train.shape[0]):
    num_aug = 0
    x = X_train[i]  # original image shape: (3,32,32)
    x = x.reshape((1,) + x.shape)  # reshape for datagen: (1,3,32,32)

    # generate augmented images
    for x_aug in datagen.flow(x,
                              batch_size=1,
                              save_to_dir='preview',
                              save_prefix='cifar',
                              save_format='jpeg'):

        if num_aug >= NUM_TO_AUGMENT:
            break
        # else
        xtas.append(x_aug[0])  # append augmented image
        num_aug += 1
#=============================================================================================
# OHE
Y_train = to_categorical(y_train, N_CLASSES)
Y_test = to_categorical(y_test, N_CLASSES)

# float and normalization
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
#=============================================================================================
# network
model = Sequential()  # TODO: explicitly define shape of inputs using Input() layer

model.add(Conv2D(32, (3,3), padding='same', input_shape=(IMG_ROWS, IMG_COLS, IMG_CHANNELS)))
model.add(Activation('relu'))
model.add(Conv2D(32, (3,3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3,3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(64, (3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))

model.add(Dense(N_CLASSES))
model.add(Activation('softmax'))
#=============================================================================================
model.summary()

# train
model.compile(loss='categorical_crossentropy', optimizer=OPTIM, metrics=['accuracy'])
```

```
#model.fit(X_train, Y_train, batch_size=BATCH_SIZE, epochs=N_EPOCHS, validation_split=VALIDATION_SI

datagen.fit(X_train)

history = model.fit(
    datagen.flow(X_train, Y_train, batch_size=BATCH_SIZE),
    steps_per_epoch=X_train.shape[0] // BATCH_SIZE,
    epochs=N_EPOCHS,
    verbose=VERBOSE
)

score = model.evaluate(X_test, Y_test, batch_size=BATCH_SIZE, verbose=VERBOSE)
print("Test score:", score[0])
print('Test accuracy:', score[1])
#============================================================================
# save model
model_json = model.to_json()
open('cifar10_architecture.json', 'w').write(model_json)
model.save_weights('cifar10_weights.weights.h5', overwrite=True)
```

```
X_train shape: (50000, 32, 32, 3)
50000 train samples
10000 test samples
Augmenting training set images...
```

Model: "sequential_7"

Layer (type)	Output Shape	Param #
conv2d_13 (Conv2D)	(None, 32, 32, 32)	896
activation_25 (Activation)	(None, 32, 32, 32)	0
conv2d_14 (Conv2D)	(None, 32, 32, 32)	9,248
activation_26 (Activation)	(None, 32, 32, 32)	0
max_pooling2d_9 (MaxPooling2D)	(None, 16, 16, 32)	0
dropout_15 (Dropout)	(None, 16, 16, 32)	0
conv2d_15 (Conv2D)	(None, 16, 16, 64)	18,496
activation_27 (Activation)	(None, 16, 16, 64)	0
conv2d_16 (Conv2D)	(None, 14, 14, 64)	36,928
activation_28 (Activation)	(None, 14, 14, 64)	0
max_pooling2d_10 (MaxPooling2D)	(None, 7, 7, 64)	0
dropout_16 (Dropout)	(None, 7, 7, 64)	0
flatten_6 (Flatten)	(None, 3136)	0
dense_12 (Dense)	(None, 512)	1,606,144
activation_29 (Activation)	(None, 512)	0
dropout_17 (Dropout)	(None, 512)	0
dense_13 (Dense)	(None, 10)	5,130
activation_30 (Activation)	(None, 10)	0

Total params: 1,676,842 (6.40 MB)

Trainable params: 1,676,842 (6.40 MB)

Non-trainable params: 0 (0.00 B)

```
Epoch 1/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 62s 153ms/step - accuracy: 0.2133 - loss: 2.1145
Epoch 2/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 0s 60us/step - accuracy: 0.4297 - loss: 0.8391
Epoch 3/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 54s 136ms/step - accuracy: 0.3600 - loss: 1.7633
Epoch 4/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 0s 27us/step - accuracy: 0.4297 - loss: 0.7570
Epoch 5/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 55s 139ms/step - accuracy: 0.4194 - loss: 1.5985
Epoch 6/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 0s 23us/step - accuracy: 0.4609 - loss: 0.7365
Epoch 7/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 56s 143ms/step - accuracy: 0.4597 - loss: 1.5078
Epoch 8/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 0s 26us/step - accuracy: 0.4609 - loss: 0.7092
Epoch 9/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 57s 146ms/step - accuracy: 0.4840 - loss: 1.4326
Epoch 10/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 0s 37us/step - accuracy: 0.5703 - loss: 0.6354
Epoch 11/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 56s 142ms/step - accuracy: 0.5042 - loss: 1.3792
Epoch 12/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 0s 33us/step - accuracy: 0.6094 - loss: 0.6136
Epoch 13/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 53s 134ms/step - accuracy: 0.5259 - loss: 1.3303
Epoch 14/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 0s 21us/step - accuracy: 0.4531 - loss: 0.7287
Epoch 15/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 52s 133ms/step - accuracy: 0.5395 - loss: 1.2900
Epoch 16/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 0s 23us/step - accuracy: 0.5078 - loss: 0.7048
Epoch 17/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 52s 131ms/step - accuracy: 0.5539 - loss: 1.2604
Epoch 18/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 0s 23us/step - accuracy: 0.5078 - loss: 0.6550
Epoch 19/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 52s 132ms/step - accuracy: 0.5597 - loss: 1.2302
Epoch 20/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 0s 28us/step - accuracy: 0.5156 - loss: 0.6612
Epoch 21/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 52s 133ms/step - accuracy: 0.5731 - loss: 1.1988
Epoch 22/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 0s 23us/step - accuracy: 0.5391 - loss: 0.6867
Epoch 23/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 52s 132ms/step - accuracy: 0.5811 - loss: 1.1873
Epoch 24/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 0s 57us/step - accuracy: 0.6484 - loss: 0.5066
Epoch 25/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 54s 138ms/step - accuracy: 0.5914 - loss: 1.1597
Epoch 26/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 0s 53us/step - accuracy: 0.6172 - loss: 0.5815
Epoch 27/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 54s 138ms/step - accuracy: 0.5965 - loss: 1.1473
Epoch 28/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 0s 60us/step - accuracy: 0.6406 - loss: 0.5372
Epoch 29/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 53s 134ms/step - accuracy: 0.6027 - loss: 1.1333
Epoch 30/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 0s 26us/step - accuracy: 0.5312 - loss: 0.5939
Epoch 31/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 55s 140ms/step - accuracy: 0.6059 - loss: 1.1211
```

```
Epoch 32/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 0s 26us/step - accuracy: 0.6328 - loss: 0.5718
Epoch 33/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 53s 135ms/step - accuracy: 0.6120 - loss: 1.0978
Epoch 34/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 0s 23us/step - accuracy: 0.6406 - loss: 0.4971
Epoch 35/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 53s 134ms/step - accuracy: 0.6105 - loss: 1.1016
Epoch 36/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 0s 26us/step - accuracy: 0.5391 - loss: 0.6426
Epoch 37/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 54s 138ms/step - accuracy: 0.6205 - loss: 1.0867
Epoch 38/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 0s 23us/step - accuracy: 0.6094 - loss: 0.5599
Epoch 39/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 52s 132ms/step - accuracy: 0.6261 - loss: 1.0740
Epoch 40/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 0s 26us/step - accuracy: 0.7031 - loss: 0.3798
Epoch 41/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 54s 137ms/step - accuracy: 0.6252 - loss: 1.0721
Epoch 42/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 0s 23us/step - accuracy: 0.6172 - loss: 0.6001
Epoch 43/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 53s 134ms/step - accuracy: 0.6316 - loss: 1.0687
Epoch 44/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 0s 69us/step - accuracy: 0.6328 - loss: 0.4854
Epoch 45/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 59s 150ms/step - accuracy: 0.6329 - loss: 1.0626
Epoch 46/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 0s 44us/step - accuracy: 0.6641 - loss: 0.4749
Epoch 47/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 63s 160ms/step - accuracy: 0.6383 - loss: 1.0404
Epoch 48/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 0s 26us/step - accuracy: 0.6016 - loss: 0.5679
Epoch 49/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 60s 151ms/step - accuracy: 0.6384 - loss: 1.0424
Epoch 50/50
390/390 ━━━━━━━━━━━━━━━━━━━━ 0s 52us/step - accuracy: 0.6719 - loss: 0.4771
79/79 ━━━━━━━━━━━━━━━━━━━━ 3s 37ms/step - accuracy: 0.7279 - loss: 0.8241
Test score: 0.8258897662162781
Test accuracy: 0.7271000146865845
```

In [14]:

```python
""" OPTIONAL PREDICTION """
import numpy as np
#import scipy.misc
from PIL import Image  # Pillow
from keras.models import model_from_json
from keras.optimizers import SGD

#load model
model_architecture = 'cifar10_architecture.json'
model_weights = 'cifar10_weights.weights.h5'
model = model_from_json(open(model_architecture).read())
model.load_weights(model_weights)

#load images
img_names = ['cat-1.jpg', 'cat-2.jpg', 'dog-1.jpg', 'dog-2.jpg']

imgs = [
    np.asarray(Image.open(img_name).resize((32,32)), dtype=np.float32)
    for img_name in img_names
]
```

```
"""
imgs = [
    np.transpose(scipy.misc.imresize(scipy.misc.imread(img_name), (32,32)),
        (1,0,2)).astype('float32')
    for img_name in img_names
]
"""

# normalize
imgs = np.array(imgs) / 255

# train
optim = SGD()
model.compile(loss='categorical_crossentropy', optimizer=optim, metrics=['accuracy'])

# predict
predictions = model.predict(imgs)
predicted_classes = np.argmax(predictions, axis=1)  # returns indices of max values (highest proba
print(predicted_classes)

#=======================================
# RESULTS = 75% Accuracy
#=======================================
"""
[ 7, 3, 5, 5 ]
cat-1 : 7-horse X
cat-2 : 3-cat
dog-1 : 5-dog
dog-2 : 5-dog
"""
pass
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 90ms/step
[7 3 5 5]
```

AI: Privacy & Ethics

When it comes to ML (machine learning) algorithms like the CDNN (convolutional deep neural network) created for the CIFAR-10 image dataset, animals aren't the only image types that may be used. Biases in the training data reinforce systematic bias. For example, people of color can be more difficult to detect and classify by self-driving cars that are trained to recognize pedestrians. This type of prejudice bias is often found in sentiment analysis ML algorithms that are trained to detect emotional or subjective sentiment or text (Xiang, 2019).

After a sufficient amount of training using a large collection of diverse and unbiased data, a CDNN can be used to distinguish people's faces. Clearview developed a facial recognition system that matches an uploaded photo of a person to show publicly available images from millions of websites. Their *Smartchekr* app is used by over 600 law enforcement agencies to solve crimes like shoplifting, identity theft, credit card fraud, murder, and child sexual exploitation (Clearview AI, n.d.). However, the company only claims to have a 30-60% hit rate success and hasn't had false positives tested by the NIST (National Institute of Standards & Technology) - the defacto leader in industry testing for such concerns. Surprising as it may (or may not) seem, the company monitors who law enforcement is looking for as well. Depending on training data, the same baseline ML model can lead to an elegant, formal, and poetic AI to a sexist, racist, disrespectful one (Silipo, 2020). China's government has been known to track and racially profile Muslim Uigher minorities, with

approximately 1 million of them believed to be in internment camps (Hailweil, 2020).

In addition to these ethical concerns, AI brings with it a cornucopia of privacy and safety concerns. Risks include the reidentification of PII (personally identifiable information) or other sensitive information (Dorschel, 2019), leading to privacy implications or even identity theft. Biased machines can lead to discrimination in job hiring, law enforcement, loan approval, and other areas of significance. It is concerning to think that an app like Clearview's can be integrated into an AR (augmented reality) headset and, in almost real-time, identify anyone who happens to walk by or sit in a coffee shop near the wrong person.

Thankfully, there are entities out there attempting to regulate this type of technology, as well as provide guidelines and frameworks for safe implementation, such as GDPR (General Data Protection Regulation), IEEE (Institute of Electrical & Electronics Engineers), and AC (Association for Computer Machinery) (Stahl & Wright, 2018). RRI (responsible research and innovation), "ethics by design", and "right to be forgotten" are approaches that are recommended to follow. AI engineers must make sure to avoid making any unknown assumptions during development, and all data must be cleaned and stripped of any biases that may exist. These are factors that affect all ML models, including the CDNN I created for this assignment. Be safe. And thanks for reading!

References:

Clearview AI. (n.d.). *Plans*. https://www.clearview.ai/plans.

Dorschel, Arianna. (2019, April 24). Rethinking Data Privacy: The Impact of Machine Learning. Luminovo. https://medium.com/luminovo/data-privacy-in-machine-learning-a-technical-deep-dive-f7f0365b1d60.

Heilweil, Rebecca. (2020, Feb. 18). *Why algorithms can be racist and sexist*. Vox. https://www.vox.com/recode/2020/2/18/21121286/algorithms-bias-discrimination-facial-recognition-transparency.

Silipo, Rosaria. (2020, March 3). *How to keep bias out of your AI models*. Customer Think. https://customerthink.com/how-to-keep-bias-out-of-your-ai-models/.

Stahl, Bernd C. & Wright, David. (2018, May/June) Ethics and Privacy in AI and Big Data: Implementing Responsible Research and Innovation. IEEE Security & Privacy (Volume: 16, Issue: 3). https://ieeexplore-ieee-org.ezproxy.snhu.edu/document/8395078.

Xiang, Mark. (2019). *Human Bias in Machine Learning: What it means in our modern big data world*. Towards Data Science. https://towardsdatascience.com/bias-what-it-means-in-the-big-data-world-6e64893e92a1.

In [ ]: