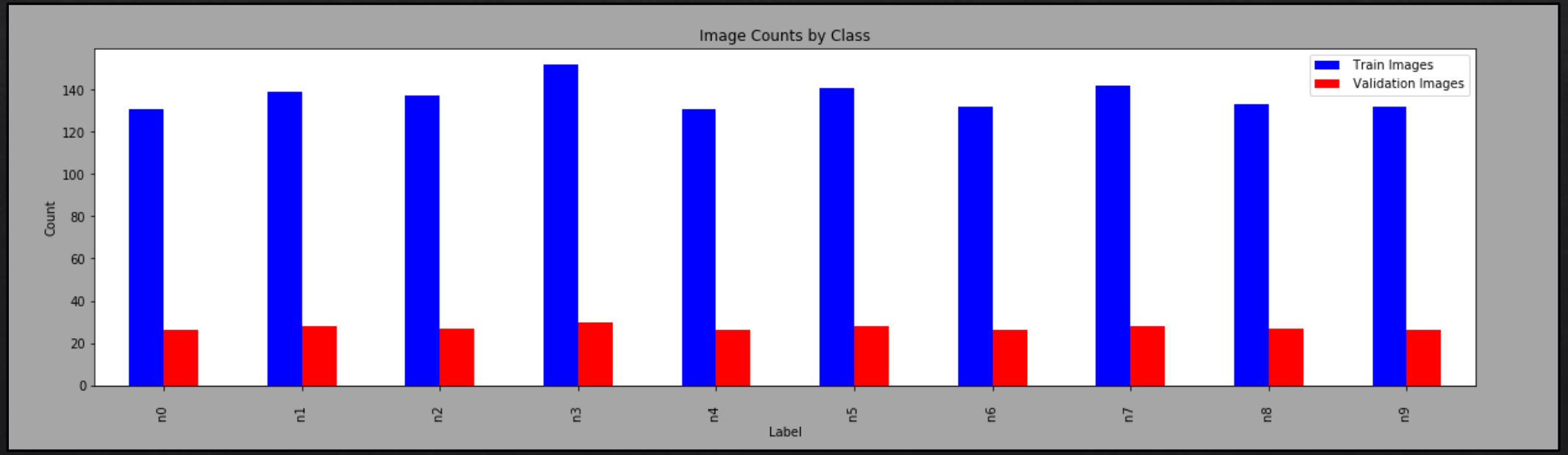
Final Capstone: Image Classification of 10 Monkey Species

By Jacob Knopping

About the Data

- "10 Monkey Species" image dataset (Kaggle)
- Images >= 400 x 300 px, JPEG
- Consists of two folders: training and validation
  - Each folder contains 10 subfolders, each corresponding to a monkey species



n0

n1

n2

n3

n4

n5

n6

n7

n8

n9

	Label	Latin Name	Common Name	Train Images	Validation Images
0	n0	alouatta_palliata\t	mantled_howler	131	26
1	n1	erythrocebus_patas\t	patas_monkey	139	28
2	n2	cacajao_calvus\t	bald_uakari	137	27
3	n3	macaca_fuscata\t	japanese_macaque	152	30
4	n4	cebuella_pygmea\t	pygmy_marmoset	131	26
5	n5	cebus_capucinus\t	white_headed_capuchin	141	28
6	n6	mico_argentatus\t	silvery_marmoset	132	26
7	n7	saimiri_sciureus\t	common_squirrel_monkey	142	28
8	n8	aotus_nigriceps\t	black_headed_night_monkey	133	27
9	n9	trachypithecus_johnii	nilgiri_langur	132	26

Project Goal

- Build a high-performance convolutional neural network (CNN) model for image classification of 10 monkey species
- In order to accomplish this goal, multiple models will be trained and evaluated
  - Build a CNN without using pre-trained models
  - Pre-trained models (Keras Applications)
    - Xception
    - InceptionResNetV2

Project Significance

- Image classification can be invaluable for wildlife conservation efforts
- Using imagery data with motion sensitive cameras = minimally invasive approach to estimation population trends in animals over time
- Drawback: human labor is currently primary means to recognize and count images
  - Impractical, impedes progress of ecological studies
- From Scientific Reports: "Deep-learning methods have revolutionized our ability to train digital computers to recognize all kinds of objects from imagery data including…wildlife species. [Deep learning/image recognition techniques] **may significantly increase the efficiency of associated ecological studies**"

Image Processing and Data Augmentation

- Keras Image Preprocessing makes this easy
  - ImageDataGenerator, flow_from_directory
- Generate training and validation data → usable format for building CNN models

```
# Establish constants
img_size = 224
batch_size = 8

# Keras ImageDataGenerator
train_datagen = ImageDataGenerator(rescale=1. / 255,
                                   rotation_range = 90,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   brightness_range=[0.75,1.0],
                                   horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1. / 255)

train_generator = train_datagen.flow_from_directory(train_dir,
                                                    target_size=(img_size, img_size),
                                                    batch_size=batch_size,
                                                    shuffle=True,
                                                    class_mode='categorical')

validation_generator = test_datagen.flow_from_directory(test_dir,
                                                        target_size=(img_size, img_size),
                                                        batch_size=batch_size,
                                                        shuffle=False,
                                                        class_mode='categorical')
```

First CNN Model

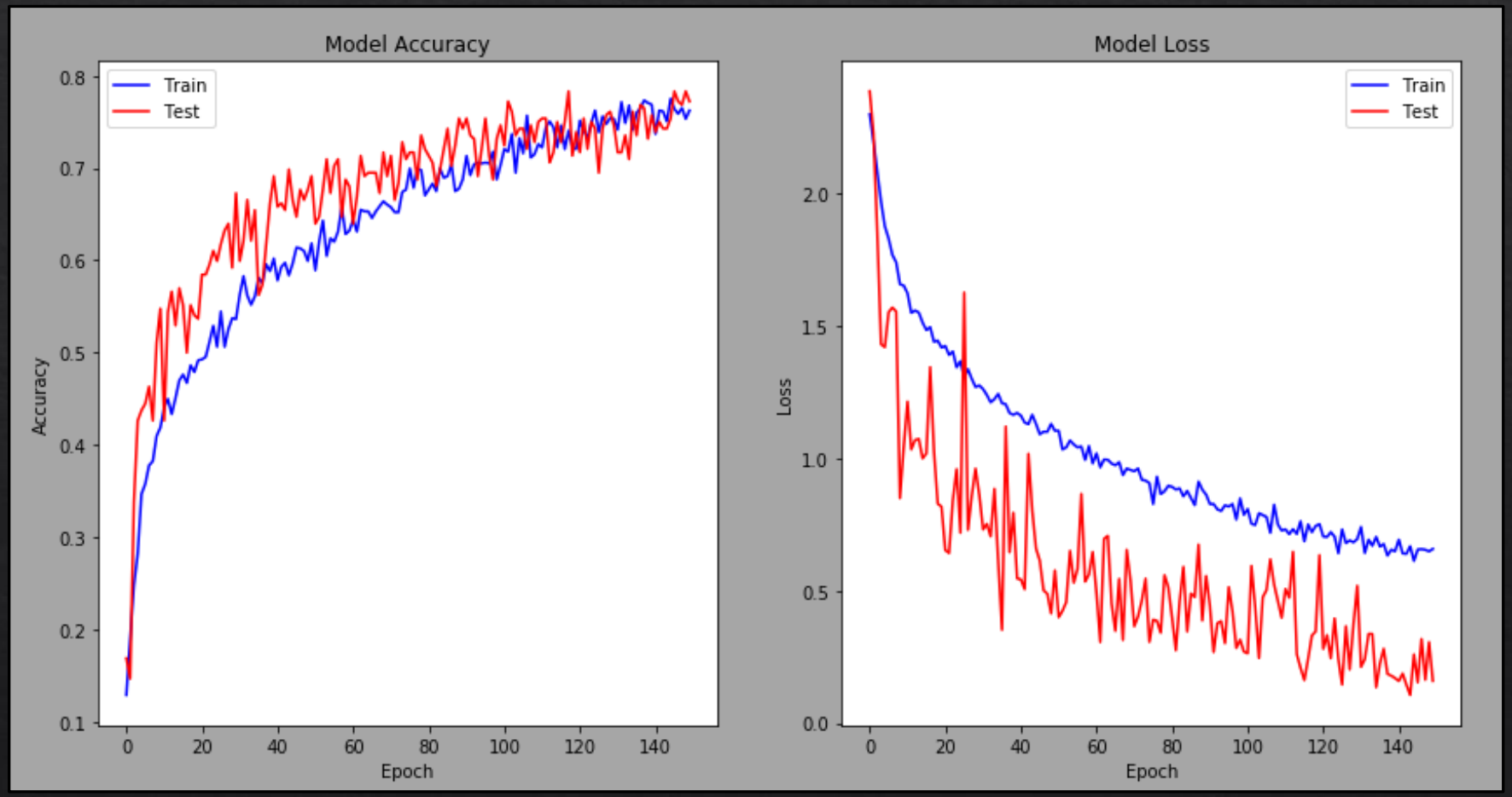
- Built with simplicity in mind: less parameters, will train quickly
  - Multiple Conv2D/MaxPooling2D layers to reduce complexity
- 462, 314 parameters (approx. 22 million, 55 million for next two)
- ~ 90 sec/epoch, 150 epochs

```
# Compile
cnn_model.compile(loss=keras.losses.categorical_crossentropy,
                  optimizer=Adam(lr=0.0001),
                  metrics=['accuracy'])
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 222, 222, 32)      896
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 111, 111, 32)      0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 109, 109, 64)      18496
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 54, 54, 64)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 52, 52, 128)       73856
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 26, 26, 128)       0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 24, 24, 64)        73792
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 12, 12, 64)        0
_________________________________________________________________
dropout_3 (Dropout)          (None, 12, 12, 64)        0
_________________________________________________________________
flatten_2 (Flatten)          (None, 9216)              0
_________________________________________________________________
dense_3 (Dense)              (None, 32)                294944
_________________________________________________________________
dropout_4 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_4 (Dense)              (None, 10)                330
=================================================================
Total params: 462,314
Trainable params: 462,314
Non-trainable params: 0
```

First CNN Model Results

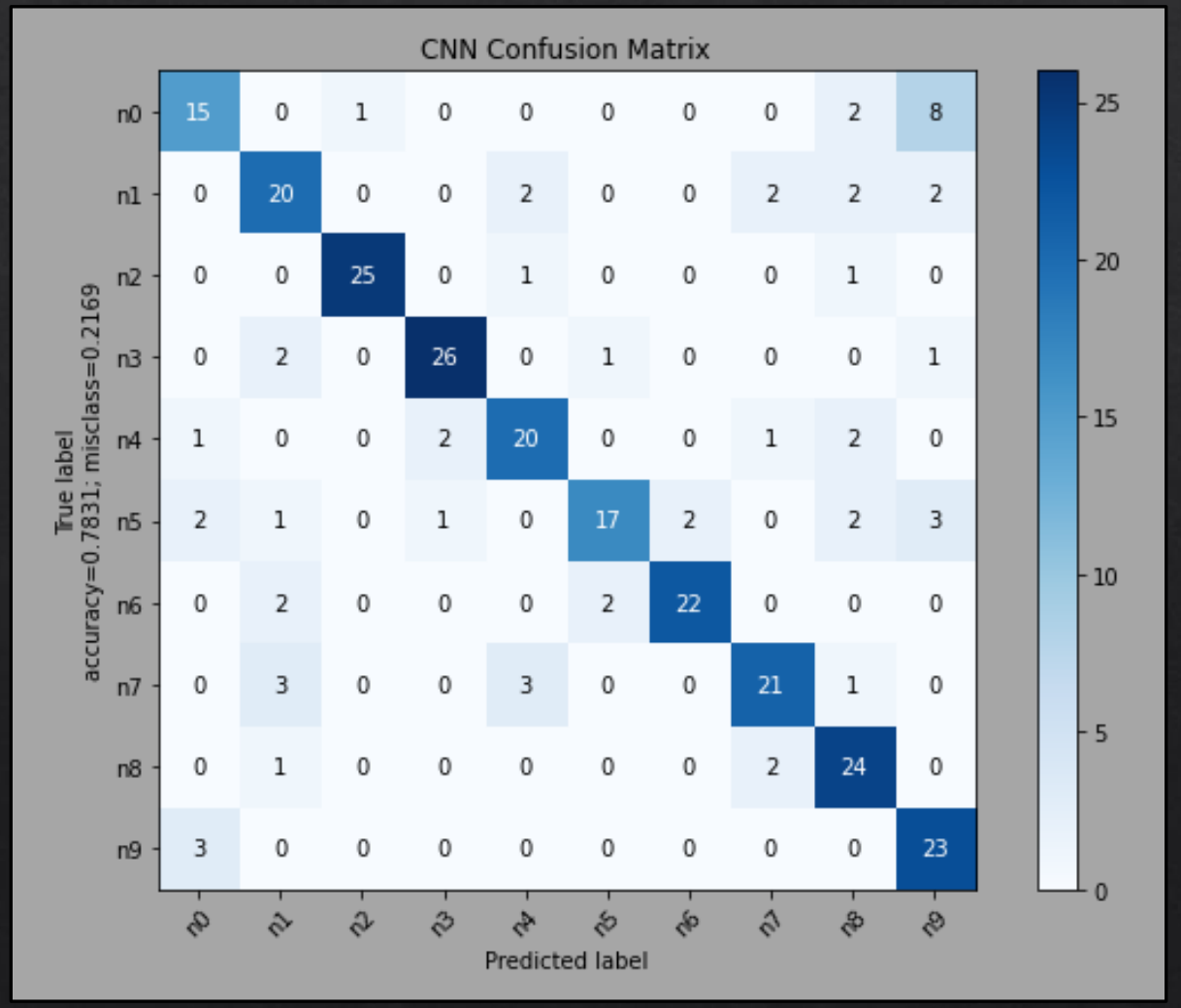
- Accuracy: 78.3%, Loss: 0.330



First CNN Model Results

- Predicting class ‘n9’ for ‘n0’ 8 times
  - Low recall
- Uneven performance overall

	precision	recall	f1-score	support
n0	0.71	0.58	0.64	26
n1	0.69	0.71	0.70	28
n2	0.96	0.93	0.94	27
n3	0.90	0.87	0.88	30
n4	0.77	0.77	0.77	26
n5	0.85	0.61	0.71	28
n6	0.92	0.85	0.88	26
n7	0.81	0.75	0.78	28
n8	0.71	0.89	0.79	27
n9	0.62	0.88	0.73	26
accuracy			0.78	272
macro avg	0.79	0.78	0.78	272
weighted avg	0.79	0.78	0.78	272



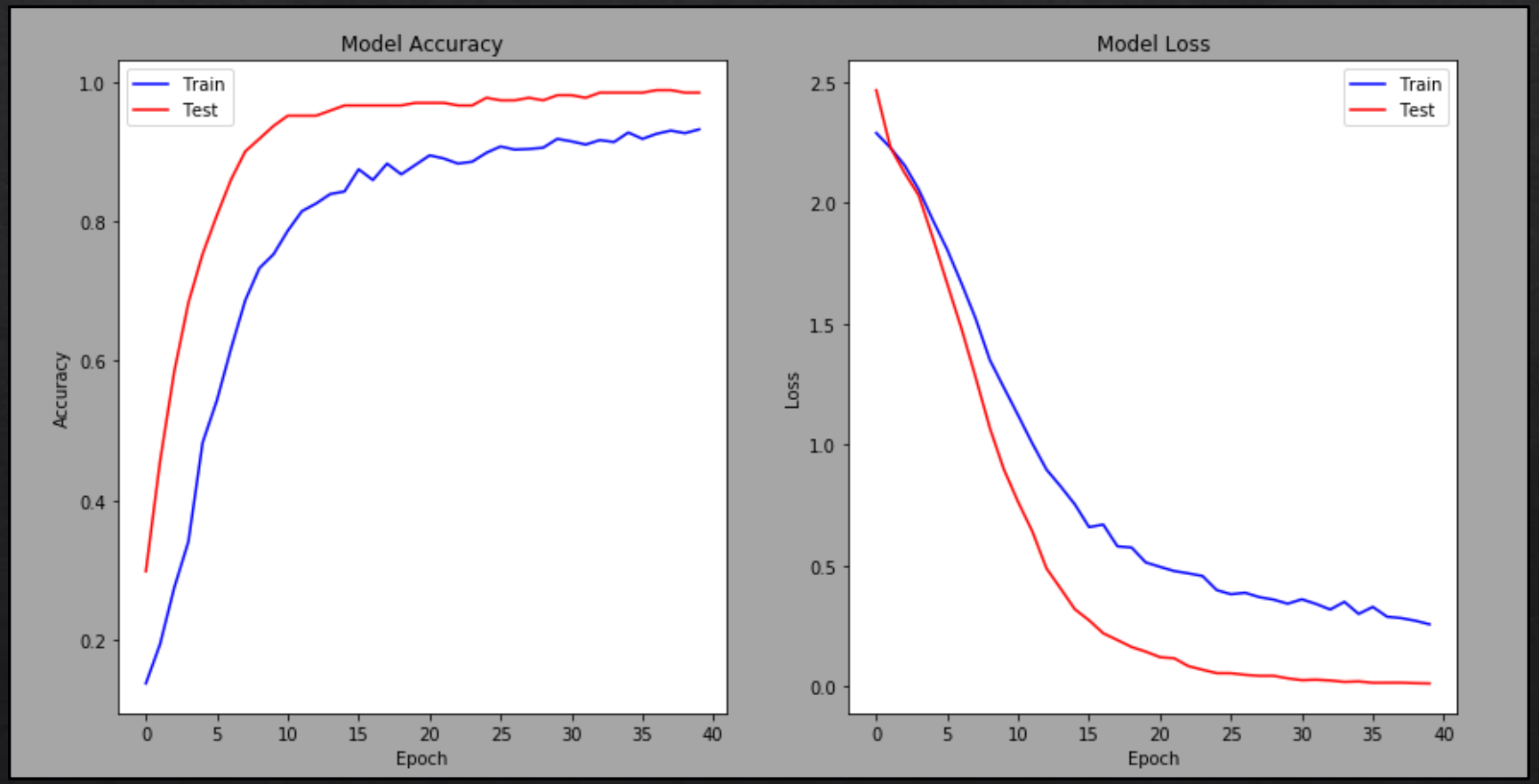
Xception Model

- High-performer on ImageNet, comparatively low # parameters
- applications.Xception(weights='imagenet', include_top=False)
  - Keras Applications
  - Used pre-trained weights, did not include top layer
- ~20.9 million parameters
- ~600 sec/epoch, 40 epochs

```
# Compile
Xception_model.compile(loss='categorical_crossentropy',
                       optimizer=SGD(lr=.0001, momentum=0.9),
                       metrics=['accuracy'])
```

Xception Model Results

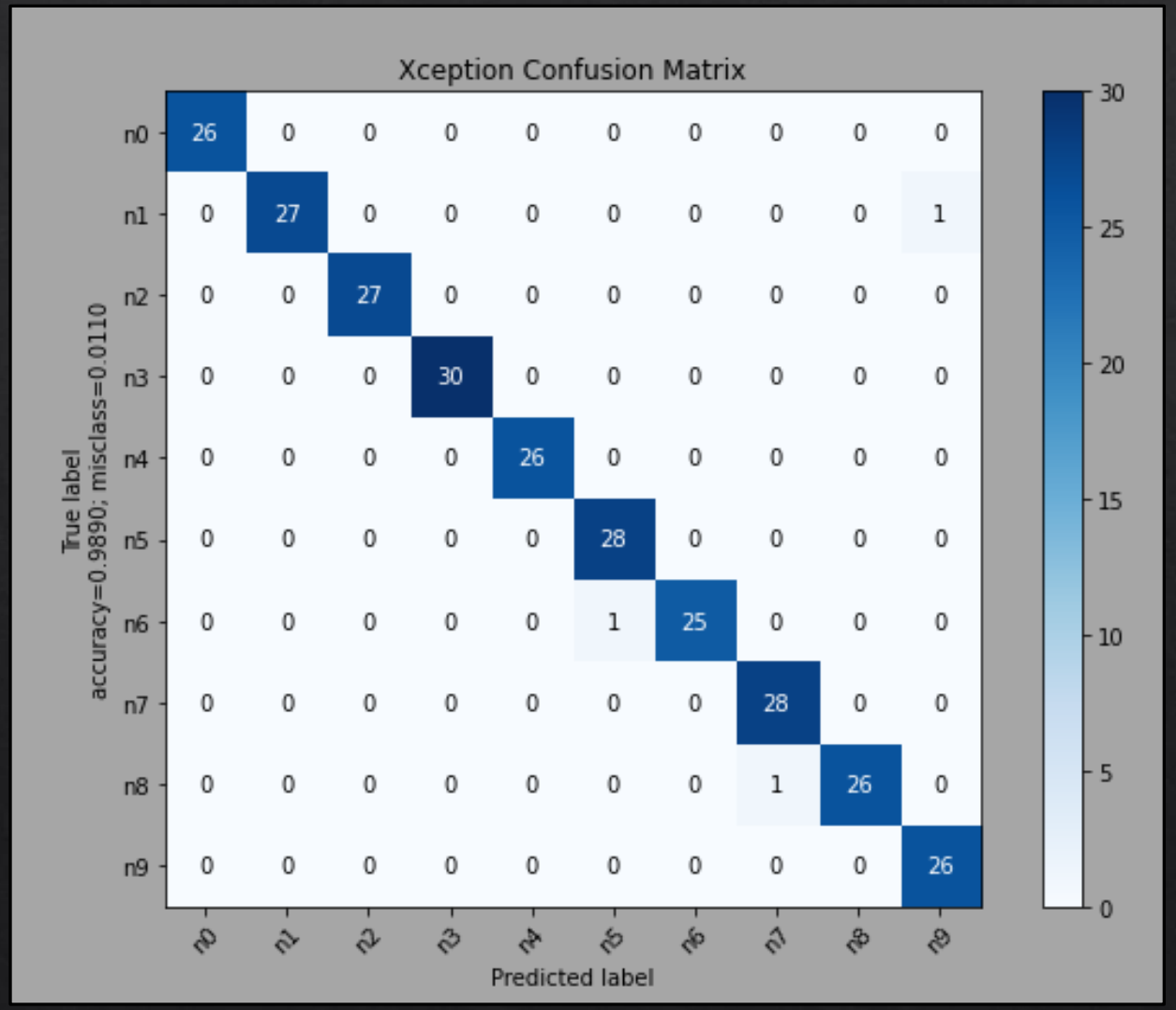
- Accuracy: 98.9%, Loss: 0.014



Xception Model Results

- Only three mislabeled classes
  - Each error for a separate class
- Recall/precision consistent

	precision	recall	f1-score	support
n0	1.00	1.00	1.00	26
n1	1.00	0.96	0.98	28
n2	1.00	1.00	1.00	27
n3	1.00	1.00	1.00	30
n4	1.00	1.00	1.00	26
n5	0.97	1.00	0.98	28
n6	1.00	0.96	0.98	26
n7	0.97	1.00	0.98	28
n8	1.00	0.96	0.98	27
n9	0.96	1.00	0.98	26
accuracy			0.99	272
macro avg	0.99	0.99	0.99	272
weighted avg	0.99	0.99	0.99	272



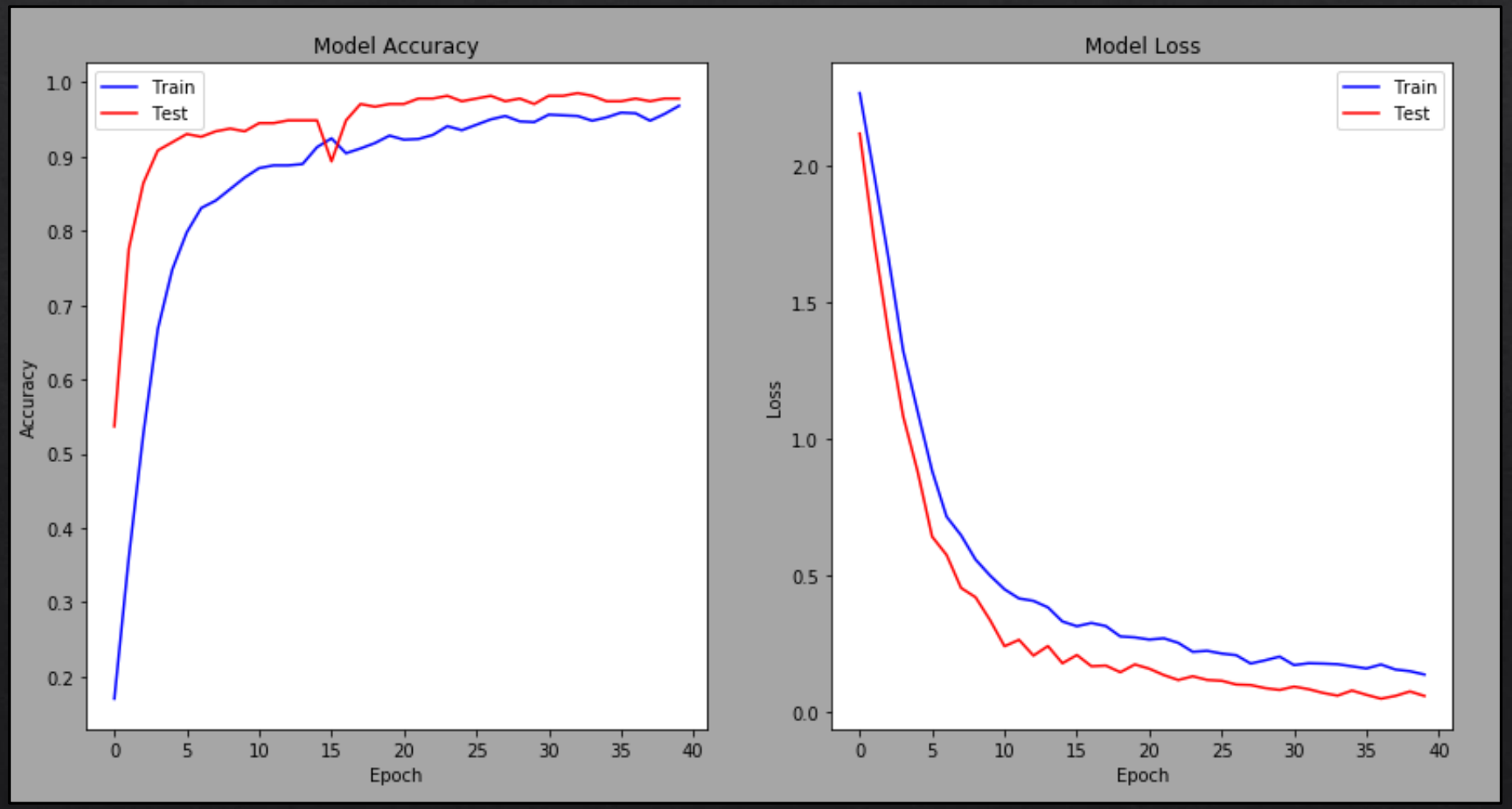
InceptionResNetV2 Model

- Slightly higher performance on ImageNet than Xception, although with a greater number of parameters
- applications.InceptionResNetV2(weights='imagenet', include_top=False)
  - Keras Applications
  - Used pre-trained weights, did not include top layer
- ~54.4 million parameters
- ~1040 secs epoch, 40 epochs

```
# Compile
resNet_model.compile(loss='categorical_crossentropy',
                     optimizer=SGD(lr=.0001, momentum=0.9),
                     metrics=['accuracy'])
```

InceptionResNetV2 Model Results

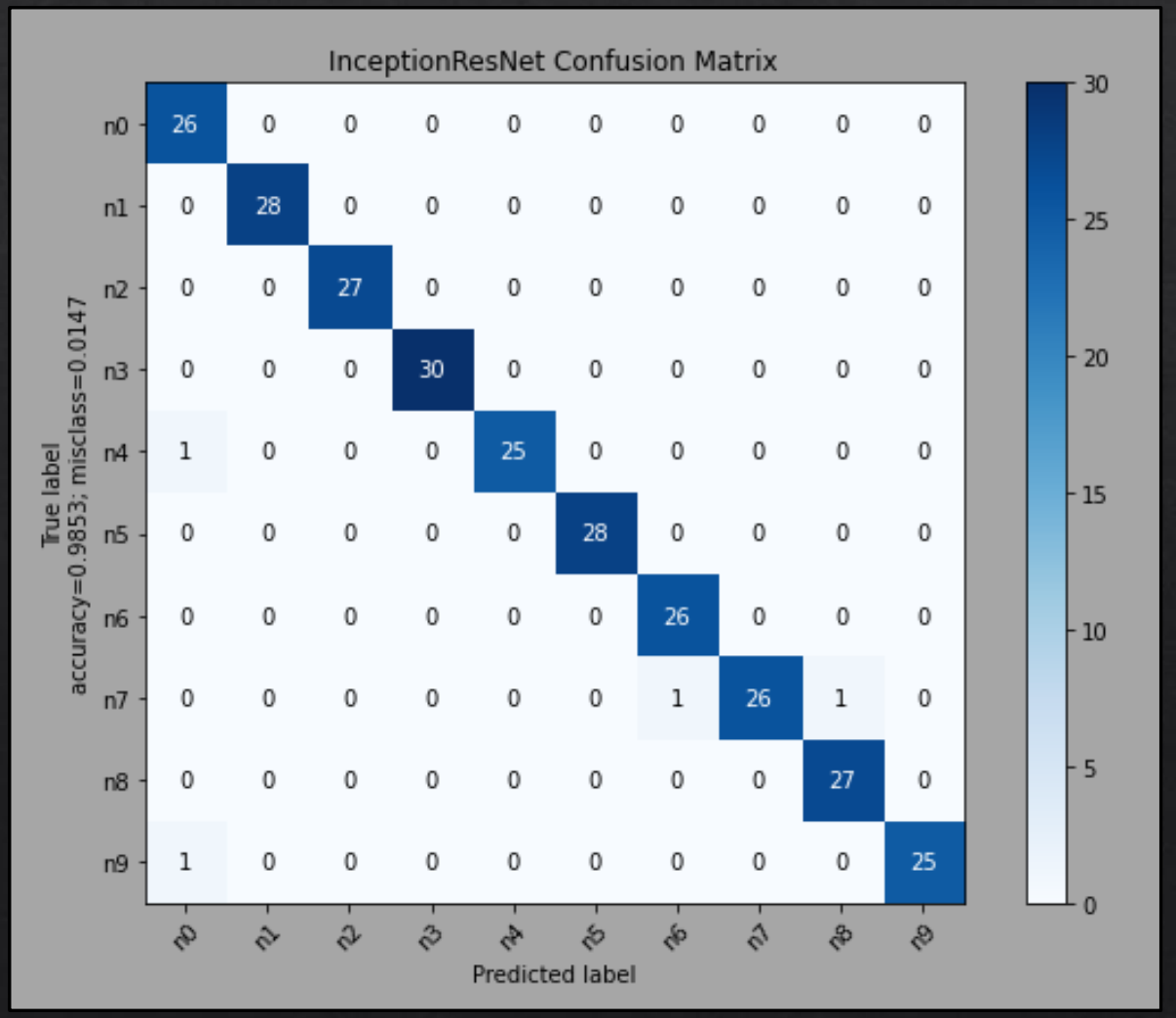
- Accuracy: 98.5%, Loss: 0.067



InceptionResNetV2 Model Results

- Only four mislabeled classes
- Precision and recall marginally less even across classes
- Comparable to Xception results

	precision	recall	f1-score	support
n0	0.93	1.00	0.96	26
n1	1.00	1.00	1.00	28
n2	1.00	1.00	1.00	27
n3	1.00	1.00	1.00	30
n4	1.00	0.96	0.98	26
n5	1.00	1.00	1.00	28
n6	0.96	1.00	0.98	26
n7	1.00	0.93	0.96	28
n8	0.96	1.00	0.98	27
n9	1.00	0.96	0.98	26
accuracy			0.99	272
macro avg	0.99	0.99	0.98	272
weighted avg	0.99	0.99	0.99	272



Conclusions

- Goal was accomplished:
  - Achieved two high performance CNN models for image classification of monkey species
- Pre-trained models both achieved 99% accuracy
- CNN model without pre-trained data (weights from ImageNet) yielded 78% accuracy on validation data
- Time cost of using more complex models proved worthwhile on this dataset
- Model could have use in ecological studies, estimating animal populations

Further Work

- InceptionResNetV2 model had higher accuracy on training data than the Xception model (~97%, vs. ~93%)
- Examining which model performs better on another dataset of the same monkey species, and whether or not this difference has any significance
- Motion capture dataset, with actual field images

QUESTIONS?