assignment06_FoxAndrea

April 25, 2021

Name: Andrea Fox
Date: April 21, 2021
Assignment: Week 6 Exercises

0.1 Assignment 6.1

Using section 5.1 in Deep Learning with Python as a guide (listing 5.3 in particular), create a

[9]:
```
#Load libraries
from keras import layers
from keras import models
from keras.datasets import mnist
from keras.utils import to_categorical
from pathlib import Path
import os
import matplotlib.pyplot as plt
import numpy as np
from keras import losses
from keras import metrics
```

[2]:
```
#5.1 Instantiating a small convnet
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

[3]:
```
#Looking at the architecture of the convnet
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
```

```
conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 3, 3, 64)          36928
=================================================================
Total params: 55,744
Trainable params: 55,744
Non-trainable params: 0
_________________________________________________________________
```

[4]:
```python
#5.2 - Adding a classifier on top of the convnet
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

[5]:
```python
#Checking architecture now to see difference
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 3, 3, 64)          36928
_________________________________________________________________
flatten (Flatten)            (None, 576)               0
_________________________________________________________________
dense (Dense)                (None, 64)                36928
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
_________________________________________________________________
```

[21]:
```python
#5.3 - Training the convnet on MNIST Images
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images = train_images.reshape((60000, 28, 28, 1))
```

```
train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype('float32') / 255

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
history = model.fit(train_images, train_labels, epochs=5, batch_size=64,␣
 ↪validation_split=0.1)
```

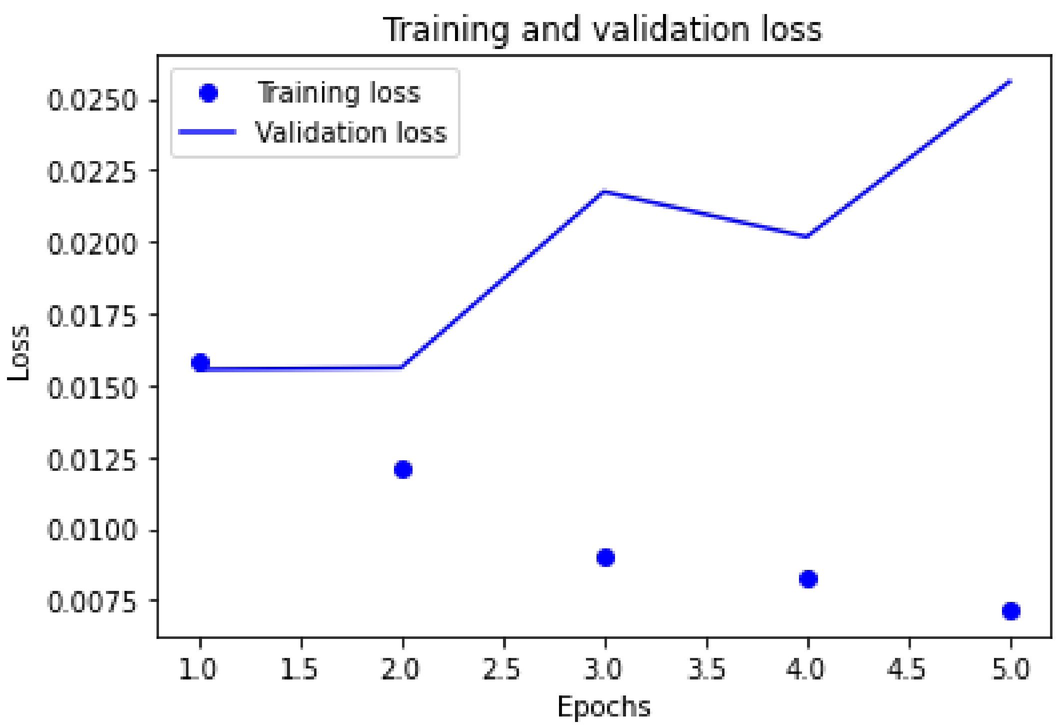
```
Epoch 1/5
844/844 [==============================] - 14s 15ms/step - loss: 0.0151 -
accuracy: 0.9955 - val_loss: 0.0156 - val_accuracy: 0.9960
Epoch 2/5
844/844 [==============================] - 12s 15ms/step - loss: 0.0109 -
accuracy: 0.9969 - val_loss: 0.0156 - val_accuracy: 0.9957
Epoch 3/5
844/844 [==============================] - 12s 15ms/step - loss: 0.0082 -
accuracy: 0.9976 - val_loss: 0.0218 - val_accuracy: 0.9940
Epoch 4/5
844/844 [==============================] - 12s 14ms/step - loss: 0.0079 -
accuracy: 0.9975 - val_loss: 0.0202 - val_accuracy: 0.9958
Epoch 5/5
844/844 [==============================] - 12s 14ms/step - loss: 0.0060 -
accuracy: 0.9982 - val_loss: 0.0256 - val_accuracy: 0.9950
```

[24]:
```
#Evaluating model on the test data
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('Test Accuracy:',test_acc)
print('Test Loss:', test_loss)
```

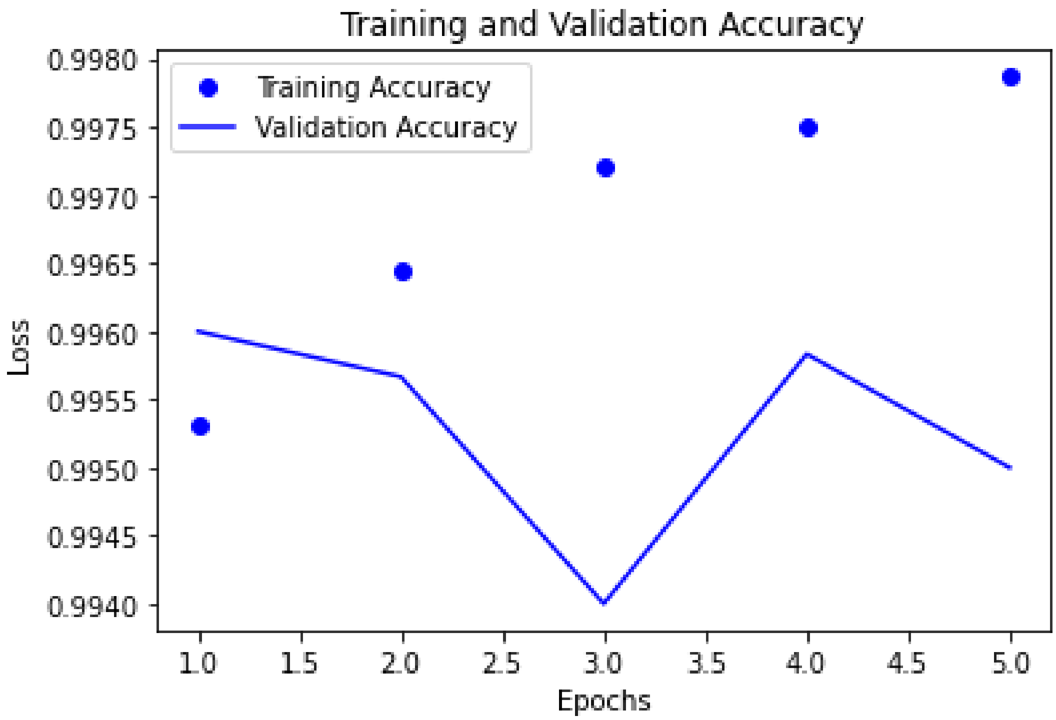
```
313/313 [==============================] - 1s 5ms/step - loss: 0.0354 -
accuracy: 0.9929
Test Accuracy: 0.992900013923645
Test Loss: 0.035402439534664154
```

[29]:
```
#Create plots
history_dict = history.history
acc_values = history_dict['accuracy']
val_acc_values = history_dict['val_accuracy']
val_loss = history_dict['loss']
val_loss_values = history_dict['val_loss']
```

```
epochs = range(1, len(accuracy) + 1)

#Plot for loss
plt.plot(epochs, val_loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss_values, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```



[28]:
```
#Plot for Accuracy
plt.plot(epochs, acc_values, 'bo', label='Training Accuracy')
plt.plot(epochs, val_acc_values, 'b', label = 'Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



0.2 Assignment 6.2

Assignment 6.2.a

Using section 5.2 in Deep Learning with Python as a guide, create a ConvNet model that classif

[49]:
```
#load libraries
from keras import layers
from keras import models
from keras import optimizers
from keras.datasets import cifar10
from matplotlib import pyplot
from keras.utils import to_categorical
from pathlib import Path
import os
import matplotlib.pyplot as plt
import numpy as np
from keras import losses
from keras import metrics
```

[39]:
```
#5.5 - Instantiating a small convnet for classification
model2 = models.Sequential()
model2.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32,␣
↪3)))
```

```
model2.add(layers.MaxPooling2D((2, 2)))
model2.add(layers.Conv2D(64, (3, 3), activation='relu'))
model2.add(layers.MaxPooling2D((2, 2)))
model2.add(layers.Conv2D(128, (3, 3), activation='relu'))
```

[40]:
```
#Looking at the architecture of the convnet
model2.summary()
```

```
Model: "sequential_4"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_18 (Conv2D)           (None, 30, 30, 32)        896

_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 15, 15, 32)        0

_________________________________________________________________
conv2d_19 (Conv2D)           (None, 13, 13, 64)        18496

_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 6, 6, 64)          0

_________________________________________________________________
conv2d_20 (Conv2D)           (None, 4, 4, 128)         73856
=================================================================
Total params: 93,248
Trainable params: 93,248
Non-trainable params: 0

_________________________________________________________________
```

[43]:
```
#Add classifier to convnet
model2.add(layers.Flatten())
model2.add(layers.Dense(512, activation='relu'))
model2.add(layers.Dense(10, activation='softmax'))
```

[44]:
```
#Second look at architecture to see difference
model2.summary()
```

```
Model: "sequential_4"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_18 (Conv2D)           (None, 30, 30, 32)        896

_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 15, 15, 32)        0

_________________________________________________________________
conv2d_19 (Conv2D)           (None, 13, 13, 64)        18496

_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 6, 6, 64)          0

_________________________________________________________________
```

```
conv2d_20 (Conv2D)           (None, 4, 4, 128)         73856

_________________________________________________________________
flatten_4 (Flatten)          (None, 2048)              0

_________________________________________________________________
dense_8 (Dense)              (None, 512)               1049088

_________________________________________________________________
dense_9 (Dense)              (None, 10)                5130
=================================================================
Total params: 1,147,466
Trainable params: 1,147,466
Non-trainable params: 0

_________________________________________________________________
```

[45]:

```
#Training the convnet on cifar10 Images
(train_images2, train_labels2), (test_images2, test_labels2) = cifar10.
 ↪load_data()

#Found this on general chat from Sam Loyd
train_images2 = train_images2.reshape((50000, 32, 32, 3))
test_images2 = test_images2.reshape((10000, 32, 32, 3))

train_labels2 = to_categorical(train_labels2)
test_labels2 = to_categorical(test_labels2)
```

[46]:

```
#5.6 - configuring the model for training. Also worked with Anna Harvey on this␣
 ↪portion
model2.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
history2 = model2.fit(train_images2, train_labels2, epochs=20, batch_size=64,␣
 ↪validation_data=(test_images2, test_labels2))
```

```
Epoch 1/20
782/782 [==============================] - 27s 33ms/step - loss: 5.9022 -
accuracy: 0.2415 - val_loss: 1.5796 - val_accuracy: 0.4406
Epoch 2/20
782/782 [==============================] - 26s 33ms/step - loss: 1.3356 -
accuracy: 0.5381 - val_loss: 1.4188 - val_accuracy: 0.5440
Epoch 3/20
782/782 [==============================] - 25s 32ms/step - loss: 1.1501 -
accuracy: 0.6117 - val_loss: 1.2844 - val_accuracy: 0.5851
Epoch 4/20
782/782 [==============================] - 25s 31ms/step - loss: 1.0808 -
accuracy: 0.6366 - val_loss: 1.4950 - val_accuracy: 0.6158
Epoch 5/20
782/782 [==============================] - 24s 31ms/step - loss: 1.0598 -
accuracy: 0.6515 - val_loss: 1.2030 - val_accuracy: 0.5845
Epoch 6/20
```

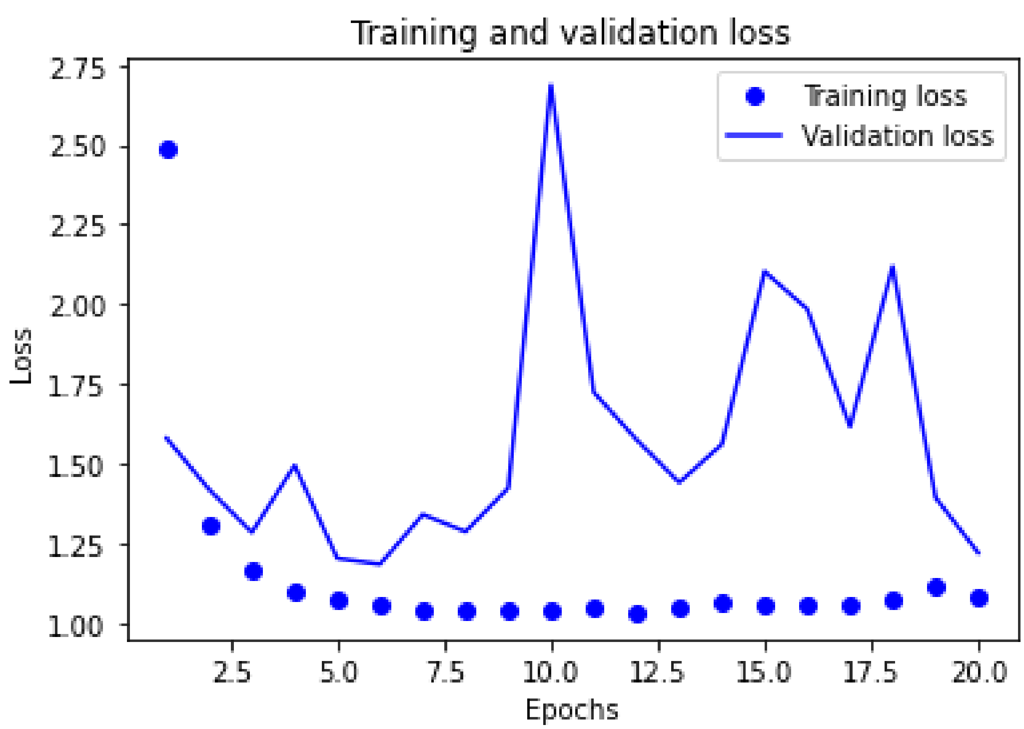
```
782/782 [==============================] - 23s 30ms/step - loss: 1.0304 -
accuracy: 0.6588 - val_loss: 1.1846 - val_accuracy: 0.6151
Epoch 7/20
782/782 [==============================] - 24s 30ms/step - loss: 1.0089 -
accuracy: 0.6719 - val_loss: 1.3406 - val_accuracy: 0.6001
Epoch 8/20
782/782 [==============================] - 24s 30ms/step - loss: 1.0084 -
accuracy: 0.6737 - val_loss: 1.2869 - val_accuracy: 0.5822
Epoch 9/20
782/782 [==============================] - 24s 30ms/step - loss: 1.0139 -
accuracy: 0.6690 - val_loss: 1.4236 - val_accuracy: 0.5944
Epoch 10/20
782/782 [==============================] - 24s 31ms/step - loss: 1.0224 -
accuracy: 0.6712 - val_loss: 2.6873 - val_accuracy: 0.4206
Epoch 11/20
782/782 [==============================] - 23s 30ms/step - loss: 1.0380 -
accuracy: 0.6694 - val_loss: 1.7243 - val_accuracy: 0.5925
Epoch 12/20
782/782 [==============================] - 23s 30ms/step - loss: 1.0256 -
accuracy: 0.6720 - val_loss: 1.5756 - val_accuracy: 0.5621
Epoch 13/20
782/782 [==============================] - 23s 30ms/step - loss: 1.0372 -
accuracy: 0.6708 - val_loss: 1.4410 - val_accuracy: 0.5844
Epoch 14/20
782/782 [==============================] - 23s 29ms/step - loss: 1.0479 -
accuracy: 0.6645 - val_loss: 1.5604 - val_accuracy: 0.5809
Epoch 15/20
782/782 [==============================] - 23s 29ms/step - loss: 1.0521 -
accuracy: 0.6640 - val_loss: 2.1028 - val_accuracy: 0.5229
Epoch 16/20
782/782 [==============================] - 23s 29ms/step - loss: 1.0554 -
accuracy: 0.6645 - val_loss: 1.9848 - val_accuracy: 0.5034
Epoch 17/20
782/782 [==============================] - 22s 29ms/step - loss: 1.0259 -
accuracy: 0.6728 - val_loss: 1.6156 - val_accuracy: 0.5410
Epoch 18/20
782/782 [==============================] - 22s 29ms/step - loss: 1.0733 -
accuracy: 0.6608 - val_loss: 2.1186 - val_accuracy: 0.4374
Epoch 19/20
782/782 [==============================] - 23s 29ms/step - loss: 1.0764 -
accuracy: 0.6621 - val_loss: 1.3937 - val_accuracy: 0.5986
Epoch 20/20
782/782 [==============================] - 22s 28ms/step - loss: 1.0463 -
accuracy: 0.6691 - val_loss: 1.2201 - val_accuracy: 0.6252
```

[72]:
```
#Evaluating model on test data
test_loss2, test_acc2 = model2.evaluate(test_images2, test_labels2)
```

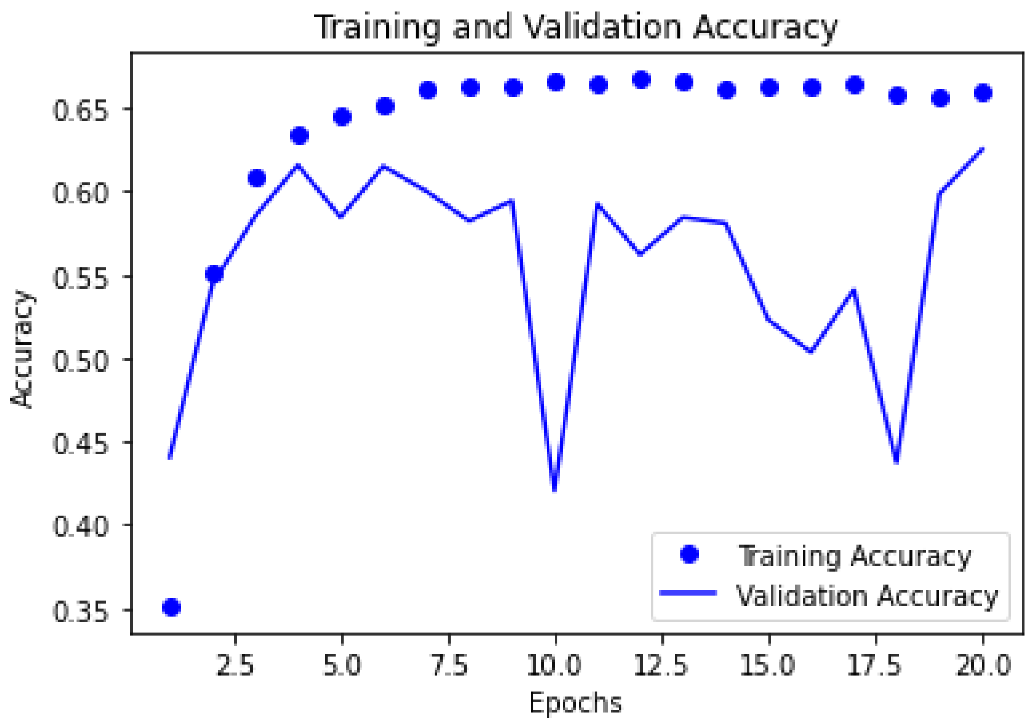
```
print('Test Accuracy:',test_acc2)
print('Test Loss:', test_loss2)
```

```
313/313 [==============================] - 2s 7ms/step - loss: 1.2201 -
accuracy: 0.6252
Test Accuracy: 0.6251999735832214
Test Loss: 1.2201370000839233
```

[52]:
```
#Create plots
history_dict2 = history2.history
acc_values2 = history_dict2['accuracy']
val_acc_values2 = history_dict2['val_accuracy']
val_loss2 = history_dict2['loss']
val_loss_values2 = history_dict2['val_loss']

epochs2 = range(1, len(val_loss2) + 1)

#Plot for loss
plt.plot(epochs2, val_loss2, 'bo', label='Training loss')
plt.plot(epochs2, val_loss_values2, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```



[54]:
```
#Plot for Accuracy
plt.plot(epochs2, acc_values2, 'bo', label='Training Accuracy')
plt.plot(epochs2, val_acc_values2, 'b', label = 'Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



Assignment 6.2.b
Using section 5.2 in Deep Learning with Python as a guide, create a ConvNet model that classif

[76]:
```
#load libraries
from keras import layers
from keras import models
from keras import optimizers
from keras.datasets import cifar10
from matplotlib import pyplot
from keras.utils import to_categorical
from pathlib import Path
```

```
import os
import matplotlib.pyplot as plt
import numpy as np
from keras import losses
from keras import metrics
from keras.preprocessing.image import ImageDataGenerator
```

[70]:
```
#Instantiating convnet for classification
model3 = models.Sequential()
model3.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32,␣
↪3)))
model3.add(layers.MaxPooling2D((2, 2)))
model3.add(layers.Dropout(0.2))
model3.add(layers.Conv2D(64, (3, 3), activation='relu'))
model3.add(layers.MaxPooling2D((2, 2)))
model3.add(layers.Dropout(0.2))
model3.add(layers.Conv2D(128, (3, 3), activation='relu'))
```

[71]:
```
#look at architecture
model3.summary()
```

```
Model: "sequential_13"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_36 (Conv2D)           (None, 30, 30, 32)        896
_________________________________________________________________
max_pooling2d_32 (MaxPooling (None, 15, 15, 32)        0
_________________________________________________________________
dropout_7 (Dropout)          (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_37 (Conv2D)           (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_33 (MaxPooling (None, 6, 6, 64)          0
_________________________________________________________________
dropout_8 (Dropout)          (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_38 (Conv2D)           (None, 4, 4, 128)         73856
=================================================================
Total params: 93,248
Trainable params: 93,248
Non-trainable params: 0
_________________________________________________________________
```

[73]:
```
#Add classifier to convnet
model3.add(layers.Flatten())
model3.add(layers.Dense(512, activation='relu'))
```

```
model3.add(layers.Dense(10, activation='softmax'))
```

[74]:
```
#See updated architecture
model3.summary()
```

```
Model: "sequential_13"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_36 (Conv2D)           (None, 30, 30, 32)        896
_________________________________________________________________
max_pooling2d_32 (MaxPooling (None, 15, 15, 32)        0
_________________________________________________________________
dropout_7 (Dropout)          (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_37 (Conv2D)           (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_33 (MaxPooling (None, 6, 6, 64)          0
_________________________________________________________________
dropout_8 (Dropout)          (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_38 (Conv2D)           (None, 4, 4, 128)         73856
_________________________________________________________________
flatten_5 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_10 (Dense)             (None, 512)               1049088
_________________________________________________________________
dense_11 (Dense)             (None, 10)                5130
=================================================================
Total params: 1,147,466
Trainable params: 1,147,466
Non-trainable params: 0
_________________________________________________________________
```

[89]:
```
#Training the convnet on cifar10 Images
(train_images3, train_labels3), (test_images3, test_labels3) = cifar10.
 ↪load_data()

#Found this on general chat from Sam Loyd
train_images3 = train_images3.reshape((50000, 32, 32, 3))
test_images3 = test_images3.reshape((10000, 32, 32, 3))

train_labels3 = to_categorical(train_labels3)
test_labels3 = to_categorical(test_labels3)
```

[90]:
```
#configuring the model for training. Also worked with Anna Harvey on this␣
 ↪portion
```

```
model3.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
#Got help from Samuel Sears on this portion
train_datagen = ImageDataGenerator(width_shift_range=0.1, height_shift_range=0.
 ↪1, horizontal_flip = True)
train_generator = train_datagen.flow(train_images3, train_labels3, batch_size =␣
 ↪64)
```

[91]:
```
#Samuel Sears said to use 200 epochs as you started seeing better results␣
 ↪around there and the plots were better
history3 = model3.fit(train_generator, epochs = 200, validation_data =␣
 ↪(test_images3, test_labels3))
```

```
Epoch 1/200
782/782 [==============================] - 54s 68ms/step - loss:
190645433183.1418 - accuracy: 0.1016 - val_loss: 1744569856.0000 - val_accuracy:
0.1000
Epoch 2/200
782/782 [==============================] - 53s 67ms/step - loss: 334387327.2031
- accuracy: 0.1025 - val_loss: 2.3098 - val_accuracy: 0.1000
Epoch 3/200
782/782 [==============================] - 53s 68ms/step - loss: 2.3290 -
accuracy: 0.1000 - val_loss: 2.3212 - val_accuracy: 0.1000
Epoch 4/200
782/782 [==============================] - 53s 68ms/step - loss: 2.3249 -
accuracy: 0.0989 - val_loss: 2.3313 - val_accuracy: 0.1000
Epoch 5/200
782/782 [==============================] - 53s 68ms/step - loss: 2.3226 -
accuracy: 0.1024 - val_loss: 2.3266 - val_accuracy: 0.1000
Epoch 6/200
782/782 [==============================] - 53s 68ms/step - loss: 2.3230 -
accuracy: 0.0981 - val_loss: 2.3162 - val_accuracy: 0.1000
Epoch 7/200
782/782 [==============================] - 53s 67ms/step - loss: 2.3206 -
accuracy: 0.1042 - val_loss: 2.3193 - val_accuracy: 0.1000
Epoch 8/200
782/782 [==============================] - 50s 63ms/step - loss: 2.3214 -
accuracy: 0.0967 - val_loss: 2.3196 - val_accuracy: 0.1000
Epoch 9/200
782/782 [==============================] - 49s 63ms/step - loss: 2.3209 -
accuracy: 0.0994 - val_loss: 2.3194 - val_accuracy: 0.1000
Epoch 10/200
782/782 [==============================] - 49s 63ms/step - loss: 2.3192 -
accuracy: 0.1013 - val_loss: 2.3192 - val_accuracy: 0.1000
Epoch 11/200
782/782 [==============================] - 50s 64ms/step - loss: 2.3191 -
```

```
accuracy: 0.0999 - val_loss: 2.3132 - val_accuracy: 0.1000
Epoch 12/200
782/782 [==============================] - 50s 63ms/step - loss: 2.3191 -
accuracy: 0.0988 - val_loss: 2.3344 - val_accuracy: 0.1000
Epoch 13/200
782/782 [==============================] - 49s 63ms/step - loss: 2.3189 -
accuracy: 0.0996 - val_loss: 2.3118 - val_accuracy: 0.1000
Epoch 14/200
782/782 [==============================] - 49s 63ms/step - loss: 2.3174 -
accuracy: 0.0966 - val_loss: 2.3418 - val_accuracy: 0.1000
Epoch 15/200
782/782 [==============================] - 49s 63ms/step - loss: 2.3156 -
accuracy: 0.1055 - val_loss: 2.3137 - val_accuracy: 0.1000
Epoch 16/200
782/782 [==============================] - 49s 63ms/step - loss: 2.3171 -
accuracy: 0.0972 - val_loss: 2.3117 - val_accuracy: 0.1000
Epoch 17/200
782/782 [==============================] - 49s 63ms/step - loss: 2.3152 -
accuracy: 0.1001 - val_loss: 2.3156 - val_accuracy: 0.1000
Epoch 18/200
782/782 [==============================] - 49s 63ms/step - loss: 2.3152 -
accuracy: 0.1013 - val_loss: 2.3172 - val_accuracy: 0.1000
Epoch 19/200
782/782 [==============================] - 50s 64ms/step - loss: 2.3149 -
accuracy: 0.0987 - val_loss: 2.3186 - val_accuracy: 0.1000
Epoch 20/200
782/782 [==============================] - 50s 63ms/step - loss: 2.3137 -
accuracy: 0.0996 - val_loss: 2.3112 - val_accuracy: 0.1000
Epoch 21/200
782/782 [==============================] - 49s 63ms/step - loss: 2.3123 -
accuracy: 0.1018 - val_loss: 2.3140 - val_accuracy: 0.1000
Epoch 22/200
782/782 [==============================] - 49s 63ms/step - loss: 2.3134 -
accuracy: 0.0959 - val_loss: 2.3106 - val_accuracy: 0.1000
Epoch 23/200
782/782 [==============================] - 50s 63ms/step - loss: 2.3127 -
accuracy: 0.0986 - val_loss: 2.3093 - val_accuracy: 0.1000
Epoch 24/200
782/782 [==============================] - 49s 63ms/step - loss: 2.3116 -
accuracy: 0.0984 - val_loss: 2.3052 - val_accuracy: 0.1000
Epoch 25/200
782/782 [==============================] - 49s 63ms/step - loss: 2.3110 -
accuracy: 0.1023 - val_loss: 2.3121 - val_accuracy: 0.1000
Epoch 26/200
782/782 [==============================] - 49s 63ms/step - loss: 2.3114 -
accuracy: 0.1009 - val_loss: 2.3129 - val_accuracy: 0.1000
Epoch 27/200
782/782 [==============================] - 50s 64ms/step - loss: 2.3122 -
```

```
accuracy: 0.1007 - val_loss: 2.3058 - val_accuracy: 0.1000
Epoch 28/200
782/782 [==============================] - 50s 64ms/step - loss: 2.3102 -
accuracy: 0.0985 - val_loss: 2.3103 - val_accuracy: 0.1000
Epoch 29/200
782/782 [==============================] - 50s 64ms/step - loss: 2.3097 -
accuracy: 0.0998 - val_loss: 2.3067 - val_accuracy: 0.1000
Epoch 30/200
782/782 [==============================] - 45s 58ms/step - loss: 2.3091 -
accuracy: 0.0993 - val_loss: 2.3076 - val_accuracy: 0.1000
Epoch 31/200
782/782 [==============================] - 42s 53ms/step - loss: 2.3078 -
accuracy: 0.1009 - val_loss: 2.3070 - val_accuracy: 0.1000
Epoch 32/200
782/782 [==============================] - 48s 61ms/step - loss: 2.3077 -
accuracy: 0.0991 - val_loss: 2.3055 - val_accuracy: 0.1000
Epoch 33/200
782/782 [==============================] - 48s 61ms/step - loss: 2.3067 -
accuracy: 0.1003 - val_loss: 2.3060 - val_accuracy: 0.1000
Epoch 34/200
782/782 [==============================] - 47s 61ms/step - loss: 2.3065 -
accuracy: 0.1025 - val_loss: 2.3049 - val_accuracy: 0.1000
Epoch 35/200
782/782 [==============================] - 47s 61ms/step - loss: 2.3065 -
accuracy: 0.1016 - val_loss: 2.3072 - val_accuracy: 0.1000
Epoch 36/200
782/782 [==============================] - 48s 61ms/step - loss: 2.3061 -
accuracy: 0.1007 - val_loss: 2.3056 - val_accuracy: 0.1000
Epoch 37/200
782/782 [==============================] - 48s 61ms/step - loss: 2.3056 -
accuracy: 0.0999 - val_loss: 2.3057 - val_accuracy: 0.1000
Epoch 38/200
782/782 [==============================] - 48s 61ms/step - loss: 2.3067 -
accuracy: 0.0965 - val_loss: 2.3039 - val_accuracy: 0.1000
Epoch 39/200
782/782 [==============================] - 47s 61ms/step - loss: 2.3054 -
accuracy: 0.1009 - val_loss: 2.3040 - val_accuracy: 0.1000
Epoch 40/200
782/782 [==============================] - 48s 61ms/step - loss: 2.3047 -
accuracy: 0.0990 - val_loss: 2.3060 - val_accuracy: 0.1000
Epoch 41/200
782/782 [==============================] - 48s 61ms/step - loss: 2.3051 -
accuracy: 0.0985 - val_loss: 2.3056 - val_accuracy: 0.1000
Epoch 42/200
782/782 [==============================] - 48s 62ms/step - loss: 2.3045 -
accuracy: 0.1021 - val_loss: 2.3033 - val_accuracy: 0.1000
Epoch 43/200
782/782 [==============================] - 48s 61ms/step - loss: 2.3044 -
```

```
accuracy: 0.1001 - val_loss: 2.3044 - val_accuracy: 0.1000
Epoch 44/200
782/782 [==============================] - 47s 60ms/step - loss: 2.3043 -
accuracy: 0.0971 - val_loss: 2.3029 - val_accuracy: 0.1000
Epoch 45/200
782/782 [==============================] - 47s 60ms/step - loss: 2.3035 -
accuracy: 0.0978 - val_loss: 2.3038 - val_accuracy: 0.1000
Epoch 46/200
782/782 [==============================] - 47s 61ms/step - loss: 2.3045 -
accuracy: 0.0975 - val_loss: 2.3030 - val_accuracy: 0.1000
Epoch 47/200
782/782 [==============================] - 47s 60ms/step - loss: 2.3034 -
accuracy: 0.1002 - val_loss: 2.3030 - val_accuracy: 0.1000
Epoch 48/200
782/782 [==============================] - 47s 60ms/step - loss: 2.3033 -
accuracy: 0.0988 - val_loss: 2.3035 - val_accuracy: 0.1000
Epoch 49/200
782/782 [==============================] - 47s 60ms/step - loss: 2.3035 -
accuracy: 0.0988 - val_loss: 2.3029 - val_accuracy: 0.1000
Epoch 50/200
782/782 [==============================] - 47s 61ms/step - loss: 2.3032 -
accuracy: 0.0984 - val_loss: 2.3028 - val_accuracy: 0.1000
Epoch 51/200
782/782 [==============================] - 48s 61ms/step - loss: 2.3029 -
accuracy: 0.1010 - val_loss: 2.3029 - val_accuracy: 0.1000
Epoch 52/200
782/782 [==============================] - 47s 60ms/step - loss: 2.3031 -
accuracy: 0.0991 - val_loss: 2.3027 - val_accuracy: 0.1000
Epoch 53/200
782/782 [==============================] - 51s 65ms/step - loss: 2.3029 -
accuracy: 0.0979 - val_loss: 2.3028 - val_accuracy: 0.1000
Epoch 54/200
782/782 [==============================] - 55s 71ms/step - loss: 2.3029 -
accuracy: 0.1000 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 55/200
782/782 [==============================] - 55s 71ms/step - loss: 2.3028 -
accuracy: 0.0988 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 56/200
782/782 [==============================] - 55s 70ms/step - loss: 2.3028 -
accuracy: 0.0984 - val_loss: 2.3027 - val_accuracy: 0.1000
Epoch 57/200
782/782 [==============================] - 55s 70ms/step - loss: 2.3028 -
accuracy: 0.0977 - val_loss: 2.3027 - val_accuracy: 0.1000
Epoch 58/200
782/782 [==============================] - 55s 70ms/step - loss: 2.3027 -
accuracy: 0.0998 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 59/200
782/782 [==============================] - 55s 71ms/step - loss: 2.3027 -
```

```
accuracy: 0.0979 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 60/200
782/782 [==============================] - 56s 71ms/step - loss: 2.3027 -
accuracy: 0.0986 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 61/200
782/782 [==============================] - 56s 71ms/step - loss: 2.3027 -
accuracy: 0.0986 - val_loss: 2.3027 - val_accuracy: 0.1000
Epoch 62/200
782/782 [==============================] - 56s 71ms/step - loss: 2.3027 -
accuracy: 0.0986 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 63/200
782/782 [==============================] - 55s 71ms/step - loss: 2.3028 -
accuracy: 0.0992 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 64/200
782/782 [==============================] - 55s 70ms/step - loss: 2.3027 -
accuracy: 0.0981 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 65/200
782/782 [==============================] - 56s 71ms/step - loss: 2.3027 -
accuracy: 0.1000 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 66/200
782/782 [==============================] - 56s 71ms/step - loss: 2.3026 -
accuracy: 0.1020 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 67/200
782/782 [==============================] - 51s 66ms/step - loss: 2.3026 -
accuracy: 0.1041 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 68/200
782/782 [==============================] - 48s 61ms/step - loss: 2.3027 -
accuracy: 0.0967 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 69/200
782/782 [==============================] - 53s 68ms/step - loss: 2.3029 -
accuracy: 0.0975 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 70/200
782/782 [==============================] - 55s 70ms/step - loss: 2.3033 -
accuracy: 0.0981 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 71/200
782/782 [==============================] - 55s 71ms/step - loss: 2.3032 -
accuracy: 0.0977 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 72/200
782/782 [==============================] - 55s 70ms/step - loss: 2.3029 -
accuracy: 0.1000 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 73/200
782/782 [==============================] - 55s 70ms/step - loss: 2.3027 -
accuracy: 0.0989 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 74/200
782/782 [==============================] - 54s 70ms/step - loss: 2.3027 -
accuracy: 0.1010 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 75/200
782/782 [==============================] - 55s 71ms/step - loss: 2.3032 -
```

```
accuracy: 0.0968 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 76/200
782/782 [==============================] - 55s 70ms/step - loss: 2.3027 -
accuracy: 0.0989 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 77/200
782/782 [==============================] - 55s 70ms/step - loss: 2.3029 -
accuracy: 0.0986 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 78/200
782/782 [==============================] - 55s 70ms/step - loss: 2.3027 -
accuracy: 0.0994 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 79/200
782/782 [==============================] - 55s 70ms/step - loss: 2.3031 -
accuracy: 0.0991 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 80/200
782/782 [==============================] - 55s 71ms/step - loss: 2.3027 -
accuracy: 0.1026 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 81/200
782/782 [==============================] - 55s 70ms/step - loss: 2.3027 -
accuracy: 0.1002 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 82/200
782/782 [==============================] - 55s 70ms/step - loss: 2.3027 -
accuracy: 0.0956 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 83/200
782/782 [==============================] - 55s 70ms/step - loss: 2.3029 -
accuracy: 0.0991 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 84/200
782/782 [==============================] - 55s 70ms/step - loss: 2.3027 -
accuracy: 0.0966 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 85/200
782/782 [==============================] - 52s 67ms/step - loss: 2.3027 -
accuracy: 0.0995 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 86/200
782/782 [==============================] - 48s 61ms/step - loss: 2.3027 -
accuracy: 0.1004 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 87/200
782/782 [==============================] - 48s 62ms/step - loss: 2.3027 -
accuracy: 0.0992 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 88/200
782/782 [==============================] - 48s 61ms/step - loss: 2.3027 -
accuracy: 0.0978 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 89/200
782/782 [==============================] - 48s 61ms/step - loss: 2.3026 -
accuracy: 0.0978 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 90/200
782/782 [==============================] - 49s 63ms/step - loss: 2.3038 -
accuracy: 0.1004 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 91/200
782/782 [==============================] - 48s 61ms/step - loss: 2.3028 -
```

```
accuracy: 0.0989 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 92/200
782/782 [==============================] - 48s 61ms/step - loss: 2.3027 -
accuracy: 0.0994 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 93/200
782/782 [==============================] - 48s 61ms/step - loss: 2.3026 -
accuracy: 0.0978 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 94/200
782/782 [==============================] - 48s 61ms/step - loss: 2.3027 -
accuracy: 0.0989 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 95/200
782/782 [==============================] - 48s 61ms/step - loss: 2.3028 -
accuracy: 0.0994 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 96/200
782/782 [==============================] - 48s 61ms/step - loss: 2.3038 -
accuracy: 0.0998 - val_loss: 2.3027 - val_accuracy: 0.1000
Epoch 97/200
782/782 [==============================] - 52s 67ms/step - loss: 2.3028 -
accuracy: 0.0989 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 98/200
782/782 [==============================] - 56s 72ms/step - loss: 2.3028 -
accuracy: 0.0970 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 99/200
782/782 [==============================] - 57s 73ms/step - loss: 2.3031 -
accuracy: 0.0989 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 100/200
782/782 [==============================] - 56s 72ms/step - loss: 2.3027 -
accuracy: 0.0958 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 101/200
782/782 [==============================] - 57s 73ms/step - loss: 2.3027 -
accuracy: 0.1009 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 102/200
782/782 [==============================] - 58s 74ms/step - loss: 2.3027 -
accuracy: 0.0994 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 103/200
782/782 [==============================] - 57s 73ms/step - loss: 2.3028 -
accuracy: 0.0952 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 104/200
782/782 [==============================] - 58s 74ms/step - loss: 2.3026 -
accuracy: 0.0986 - val_loss: 2.3027 - val_accuracy: 0.1000
Epoch 105/200
782/782 [==============================] - 58s 74ms/step - loss: 2.3029 -
accuracy: 0.0963 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 106/200
782/782 [==============================] - 64s 82ms/step - loss: 2.3033 -
accuracy: 0.0994 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 107/200
782/782 [==============================] - 57s 73ms/step - loss: 2.3030 -
```

```
accuracy: 0.0980 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 108/200
782/782 [==============================] - 57s 73ms/step - loss: 2.3029 -
accuracy: 0.0974 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 109/200
782/782 [==============================] - 58s 74ms/step - loss: 2.3027 -
accuracy: 0.0988 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 110/200
782/782 [==============================] - 58s 74ms/step - loss: 2.3027 -
accuracy: 0.1001 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 111/200
782/782 [==============================] - 58s 73ms/step - loss: 2.3027 -
accuracy: 0.0958 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 112/200
782/782 [==============================] - 57s 73ms/step - loss: 2.3028 -
accuracy: 0.0962 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 113/200
782/782 [==============================] - 57s 73ms/step - loss: 2.3088 -
accuracy: 0.0982 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 114/200
782/782 [==============================] - 48s 62ms/step - loss: 2.3027 -
accuracy: 0.0968 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 115/200
782/782 [==============================] - 48s 61ms/step - loss: 2.3029 -
accuracy: 0.0985 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 116/200
782/782 [==============================] - 55s 71ms/step - loss: 2.3027 -
accuracy: 0.0958 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 117/200
782/782 [==============================] - 57s 73ms/step - loss: 2.3069 -
accuracy: 0.1006 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 118/200
782/782 [==============================] - 58s 74ms/step - loss: 2.3027 -
accuracy: 0.0987 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 119/200
782/782 [==============================] - 56s 72ms/step - loss: 2.3028 -
accuracy: 0.0983 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 120/200
782/782 [==============================] - 57s 73ms/step - loss: 2.3028 -
accuracy: 0.0996 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 121/200
782/782 [==============================] - 57s 73ms/step - loss: 2.3028 -
accuracy: 0.1005 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 122/200
782/782 [==============================] - 57s 72ms/step - loss: 2.3027 -
accuracy: 0.0991 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 123/200
782/782 [==============================] - 49s 62ms/step - loss: 2.3037 -
```

```
accuracy: 0.1019 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 124/200
782/782 [==============================] - 48s 62ms/step - loss: 2.3029 -
accuracy: 0.1012 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 125/200
782/782 [==============================] - 49s 63ms/step - loss: 2.3027 -
accuracy: 0.1007 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 126/200
782/782 [==============================] - 48s 62ms/step - loss: 2.3027 -
accuracy: 0.1017 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 127/200
782/782 [==============================] - 48s 62ms/step - loss: 2.3027 -
accuracy: 0.0960 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 128/200
782/782 [==============================] - 56s 71ms/step - loss: 2.3028 -
accuracy: 0.1005 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 129/200
782/782 [==============================] - 57s 72ms/step - loss: 2.3027 -
accuracy: 0.0972 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 130/200
782/782 [==============================] - 57s 73ms/step - loss: 2.3027 -
accuracy: 0.0974 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 131/200
782/782 [==============================] - 57s 73ms/step - loss: 2.3030 -
accuracy: 0.0974 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 132/200
782/782 [==============================] - 58s 74ms/step - loss: 2.3027 -
accuracy: 0.0990 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 133/200
782/782 [==============================] - 59s 75ms/step - loss: 2.3027 -
accuracy: 0.1003 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 134/200
782/782 [==============================] - 58s 74ms/step - loss: 2.3026 -
accuracy: 0.0993 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 135/200
782/782 [==============================] - 51s 66ms/step - loss: 2.3027 -
accuracy: 0.1007 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 136/200
782/782 [==============================] - 56s 72ms/step - loss: 2.3031 -
accuracy: 0.0972 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 137/200
782/782 [==============================] - 55s 70ms/step - loss: 2.3027 -
accuracy: 0.0978 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 138/200
782/782 [==============================] - 54s 69ms/step - loss: 2.3027 -
accuracy: 0.0977 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 139/200
782/782 [==============================] - 53s 68ms/step - loss: 2.3027 -
```

```
accuracy: 0.1011 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 140/200
782/782 [==============================] - 54s 69ms/step - loss: 2.3027 -
accuracy: 0.1006 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 141/200
782/782 [==============================] - 56s 72ms/step - loss: 2.3027 -
accuracy: 0.0994 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 142/200
782/782 [==============================] - 54s 69ms/step - loss: 2.3027 -
accuracy: 0.0971 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 143/200
782/782 [==============================] - 54s 69ms/step - loss: 2.3026 -
accuracy: 0.1009 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 144/200
782/782 [==============================] - 64s 82ms/step - loss: 2.3036 -
accuracy: 0.1002 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 145/200
782/782 [==============================] - 53s 68ms/step - loss: 2.3028 -
accuracy: 0.0974 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 146/200
782/782 [==============================] - 53s 68ms/step - loss: 2.3027 -
accuracy: 0.0994 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 147/200
782/782 [==============================] - 53s 68ms/step - loss: 2.3027 -
accuracy: 0.1008 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 148/200
782/782 [==============================] - 52s 66ms/step - loss: 2.3027 -
accuracy: 0.0989 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 149/200
782/782 [==============================] - 50s 63ms/step - loss: 2.3028 -
accuracy: 0.0949 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 150/200
782/782 [==============================] - 49s 63ms/step - loss: 2.3027 -
accuracy: 0.0996 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 151/200
782/782 [==============================] - 49s 63ms/step - loss: 2.3027 -
accuracy: 0.0968 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 152/200
782/782 [==============================] - 50s 64ms/step - loss: 2.3027 -
accuracy: 0.0977 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 153/200
782/782 [==============================] - 49s 63ms/step - loss: 2.3026 -
accuracy: 0.1018 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 154/200
782/782 [==============================] - 48s 61ms/step - loss: 2.3027 -
accuracy: 0.0961 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 155/200
782/782 [==============================] - 46s 59ms/step - loss: 2.3027 -
```

```
accuracy: 0.0969 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 156/200
782/782 [==============================] - 46s 59ms/step - loss: 2.3027 -
accuracy: 0.1006 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 157/200
782/782 [==============================] - 52s 67ms/step - loss: 2.3027 -
accuracy: 0.1002 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 158/200
782/782 [==============================] - 54s 69ms/step - loss: 2.3027 -
accuracy: 0.0996 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 159/200
782/782 [==============================] - 55s 70ms/step - loss: 2.3027 -
accuracy: 0.0992 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 160/200
782/782 [==============================] - 55s 70ms/step - loss: 2.3027 -
accuracy: 0.1017 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 161/200
782/782 [==============================] - 55s 70ms/step - loss: 2.3027 -
accuracy: 0.0978 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 162/200
782/782 [==============================] - 60s 77ms/step - loss: 2.3027 -
accuracy: 0.1002 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 163/200
782/782 [==============================] - 55s 70ms/step - loss: 2.3027 -
accuracy: 0.0964 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 164/200
782/782 [==============================] - 58s 74ms/step - loss: 2.3050 -
accuracy: 0.0999 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 165/200
782/782 [==============================] - 64s 82ms/step - loss: 2.3039 -
accuracy: 0.0978 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 166/200
782/782 [==============================] - 64s 82ms/step - loss: 2.3027 -
accuracy: 0.0993 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 167/200
782/782 [==============================] - 64s 82ms/step - loss: 2.3027 -
accuracy: 0.0968 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 168/200
782/782 [==============================] - 64s 82ms/step - loss: 2.3026 -
accuracy: 0.0963 - val_loss: 2.3027 - val_accuracy: 0.1000
Epoch 169/200
782/782 [==============================] - 63s 81ms/step - loss: 2.3027 -
accuracy: 0.1021 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 170/200
782/782 [==============================] - 63s 81ms/step - loss: 2.3034 -
accuracy: 0.0984 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 171/200
782/782 [==============================] - 63s 80ms/step - loss: 2.3027 -
```

```
accuracy: 0.0972 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 172/200
782/782 [==============================] - 63s 80ms/step - loss: 2.3026 -
accuracy: 0.0988 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 173/200
782/782 [==============================] - 63s 80ms/step - loss: 2.3027 -
accuracy: 0.1002 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 174/200
782/782 [==============================] - 62s 79ms/step - loss: 2.3027 -
accuracy: 0.1002 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 175/200
782/782 [==============================] - 62s 80ms/step - loss: 2.3027 -
accuracy: 0.0993 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 176/200
782/782 [==============================] - 62s 80ms/step - loss: 2.3027 -
accuracy: 0.0939 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 177/200
782/782 [==============================] - 63s 80ms/step - loss: 2.3025 -
accuracy: 0.0993 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 178/200
782/782 [==============================] - 62s 79ms/step - loss: 2.3027 -
accuracy: 0.1018 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 179/200
782/782 [==============================] - 62s 80ms/step - loss: 2.3027 -
accuracy: 0.1001 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 180/200
782/782 [==============================] - 63s 81ms/step - loss: 2.3027 -
accuracy: 0.0975 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 181/200
782/782 [==============================] - 63s 80ms/step - loss: 2.3028 -
accuracy: 0.0952 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 182/200
782/782 [==============================] - 63s 81ms/step - loss: 2.3027 -
accuracy: 0.0996 - val_loss: 2.3027 - val_accuracy: 0.1000
Epoch 183/200
782/782 [==============================] - 63s 80ms/step - loss: 2.3028 -
accuracy: 0.0984 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 184/200
782/782 [==============================] - 63s 80ms/step - loss: 2.3027 -
accuracy: 0.0989 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 185/200
782/782 [==============================] - 63s 81ms/step - loss: 2.3027 -
accuracy: 0.0979 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 186/200
782/782 [==============================] - 64s 81ms/step - loss: 2.3026 -
accuracy: 0.0986 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 187/200
782/782 [==============================] - 63s 81ms/step - loss: 2.3027 -
```

```
accuracy: 0.0996 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 188/200
782/782 [==============================] - 63s 80ms/step - loss: 2.3027 -
accuracy: 0.1007 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 189/200
782/782 [==============================] - 64s 81ms/step - loss: 2.3026 -
accuracy: 0.1006 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 190/200
782/782 [==============================] - 63s 80ms/step - loss: 2.3027 -
accuracy: 0.1009 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 191/200
782/782 [==============================] - 63s 81ms/step - loss: 2.3027 -
accuracy: 0.1002 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 192/200
782/782 [==============================] - 63s 80ms/step - loss: 2.3026 -
accuracy: 0.0996 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 193/200
782/782 [==============================] - 62s 80ms/step - loss: 2.3027 -
accuracy: 0.0988 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 194/200
782/782 [==============================] - 62s 79ms/step - loss: 2.3027 -
accuracy: 0.0992 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 195/200
782/782 [==============================] - 58s 74ms/step - loss: 2.3027 -
accuracy: 0.0996 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 196/200
782/782 [==============================] - 59s 75ms/step - loss: 2.3026 -
accuracy: 0.0989 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 197/200
782/782 [==============================] - 59s 75ms/step - loss: 2.3030 -
accuracy: 0.0995 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 198/200
782/782 [==============================] - 59s 75ms/step - loss: 2.3027 -
accuracy: 0.1002 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 199/200
782/782 [==============================] - 58s 74ms/step - loss: 2.3024 -
accuracy: 0.1002 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 200/200
782/782 [==============================] - 52s 67ms/step - loss: 2.3027 -
accuracy: 0.1006 - val_loss: 2.3026 - val_accuracy: 0.1000
```

[92]:
```python
#Evaluating model on test data
test_loss3, test_acc3 = model3.evaluate(test_images3, test_labels3)
print('Test Accuracy:',test_acc3)
print('Test Loss:', test_loss3)
```

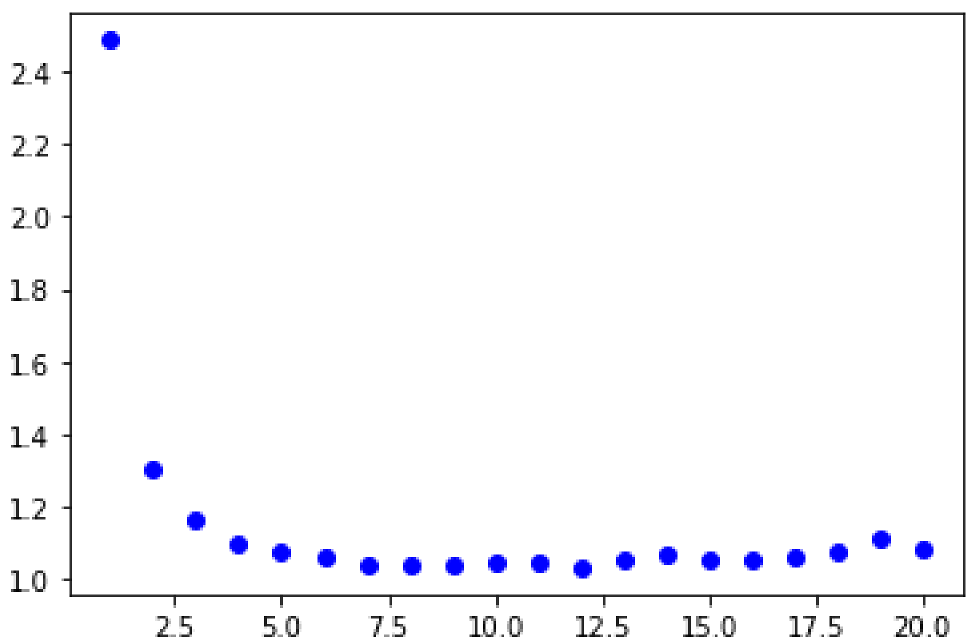
```
313/313 [==============================] - 2s 7ms/step - loss: 2.3026 -
accuracy: 0.1000
```

```
Test Accuracy: 0.10000000149011612
Test Loss: 2.302603006362915
```

[95]:
```
#Create plots
history_dict3 = history3.history
acc_values3 = history_dict3['accuracy']
val_acc_values3 = history_dict3['val_accuracy']
val_loss3 = history_dict2['loss']
val_loss_values3 = history_dict3['val_loss']

epochs3 = range(1, len(val_loss3) + 1)

#Plot for loss
plt.plot(epochs3, val_loss3, 'bo', label='Training loss')
plt.plot(epochs3, val_loss_values3, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-95-e6bce6c06167> in <module>
     10 #Plot for loss
     11 plt.plot(epochs3, val_loss3, 'bo', label='Training loss')
---> 12 plt.plot(epochs3, val_loss_values3, 'b', label='Validation loss')
     13 plt.title('Training and validation loss')
     14 plt.xlabel('Epochs')

/opt/conda/lib/python3.8/site-packages/matplotlib/pyplot.py in plot(scalex,␣
↪scaley, data, *args, **kwargs)
   2838 @_copy_docstring_and_deprecators(Axes.plot)
   2839 def plot(*args, scalex=True, scaley=True, data=None, **kwargs):
-> 2840     return gca().plot(

   2841         *args, scalex=scalex, scaley=scaley,
   2842         **({"data": data} if data is not None else {}), **kwargs)

/opt/conda/lib/python3.8/site-packages/matplotlib/axes/_axes.py in plot(self,␣
↪scalex, scaley, data, *args, **kwargs)
   1741         """
   1742         kwargs = cbook.normalize_kwargs(kwargs, mlines.Line2D)
-> 1743         lines = [*self._get_lines(*args, data=data, **kwargs)]
   1744         for line in lines:
   1745             self.add_line(line)
```

```
/opt/conda/lib/python3.8/site-packages/matplotlib/axes/_base.py in␣
↪__call__(self, data, *args, **kwargs)
    271                     this += args[0],
    272                     args = args[1:]
--> 273                 yield from self._plot_args(this, kwargs)
    274
    275     def get_next_color(self):

/opt/conda/lib/python3.8/site-packages/matplotlib/axes/_base.py in␣
↪_plot_args(self, tup, kwargs)
    397
    398         if x.shape[0] != y.shape[0]:
--> 399             raise ValueError(f"x and y must have same first dimension,␣
↪but "
    400                              f"have shapes {x.shape} and {y.shape}")
    401         if x.ndim > 2 or y.ndim > 2:

ValueError: x and y must have same first dimension, but have shapes (20,) and␣
↪(200,)
```

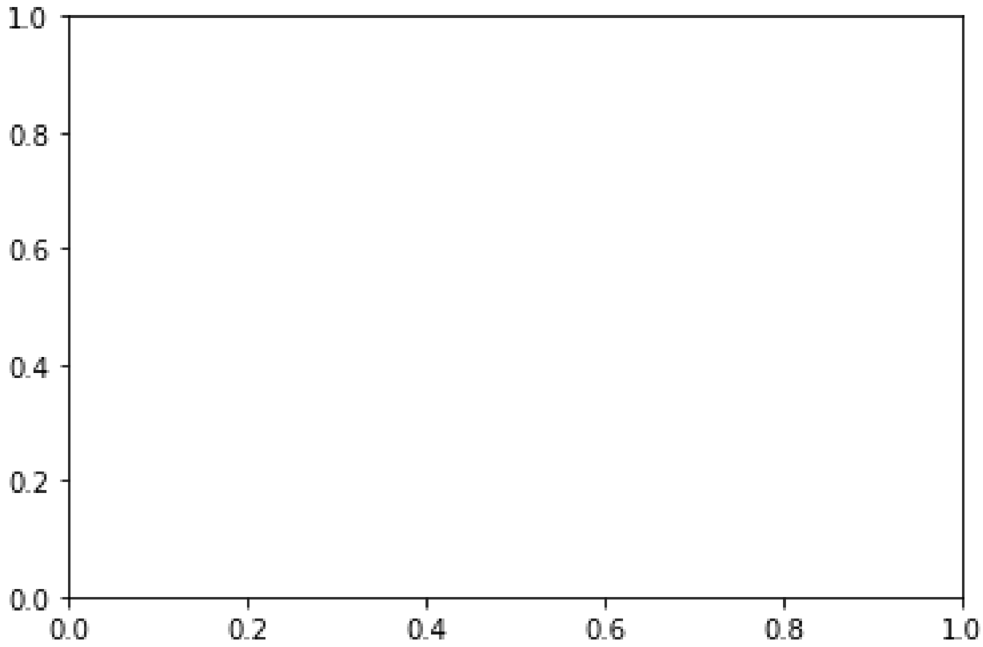


[96]:
```
#Plot for Accuracy
plt.plot(epochs3, acc_values3, 'bo', label='Training Accuracy')
plt.plot(epochs3, val_acc_values3, 'b', label = 'Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
```

```
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-96-4890c52fdf6e> in <module>
      1 #Plot for Accuracy
----> 2 plt.plot(epochs3, acc_values3, 'bo', label='Training Accuracy')
      3 plt.plot(epochs3, val_acc_values3, 'b', label = 'Validation Accuracy')
      4 plt.title('Training and Validation Accuracy')
      5 plt.xlabel('Epochs')

/opt/conda/lib/python3.8/site-packages/matplotlib/pyplot.py in plot(scalex,␣
↪scaley, data, *args, **kwargs)
   2838 @_copy_docstring_and_deprecators(Axes.plot)
   2839 def plot(*args, scalex=True, scaley=True, data=None, **kwargs):
-> 2840     return gca().plot(

   2841         *args, scalex=scalex, scaley=scaley,
   2842         **({"data": data} if data is not None else {}), **kwargs)

/opt/conda/lib/python3.8/site-packages/matplotlib/axes/_axes.py in plot(self,␣
↪scalex, scaley, data, *args, **kwargs)
   1741         """
   1742         kwargs = cbook.normalize_kwargs(kwargs, mlines.Line2D)
-> 1743         lines = [*self._get_lines(*args, data=data, **kwargs)]
   1744         for line in lines:
   1745             self.add_line(line)

/opt/conda/lib/python3.8/site-packages/matplotlib/axes/_base.py in␣
↪__call__(self, data, *args, **kwargs)
    271                 this += args[0],
    272                 args = args[1:]
--> 273             yield from self._plot_args(this, kwargs)
    274 
    275     def get_next_color(self):

/opt/conda/lib/python3.8/site-packages/matplotlib/axes/_base.py in␣
↪_plot_args(self, tup, kwargs)
    397 
    398         if x.shape[0] != y.shape[0]:
--> 399             raise ValueError(f"x and y must have same first dimension,␣
↪but "
    400                              f"have shapes {x.shape} and {y.shape}")
    401         if x.ndim > 2 or y.ndim > 2:
```

```
ValueError: x and y must have same first dimension, but have shapes (20,) and
↪(200,)
```



0.3 Assignment 6.3

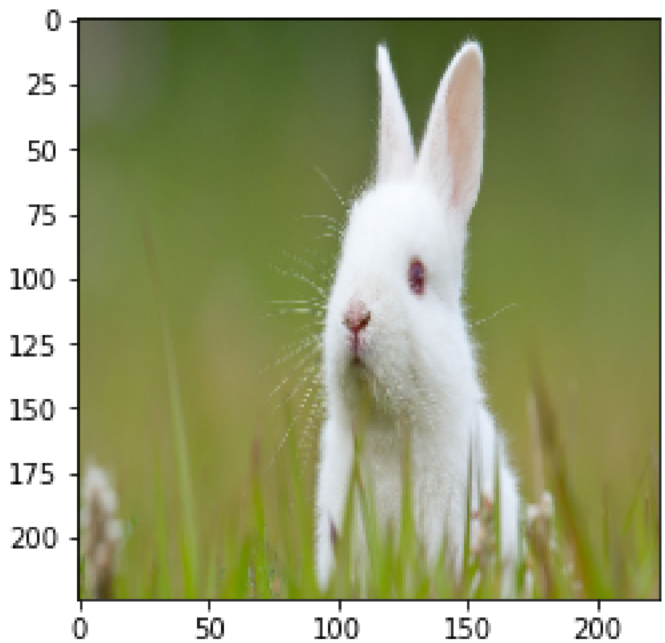
Load the ResNet50 model. Perform image classification on five to ten images of your choice. The

[31]:
```
#load libraries
import os
from pathlib import Path

#load ResNet50 libraries
import numpy as np

from keras.preprocessing.image import img_to_array
from keras.applications.resnet50 import preprocess_input
from keras.applications.imagenet_utils import decode_predictions
import matplotlib.pyplot as plt
```

[32]:
```
#set current directory and create images directory
current_dir = Path(os.getcwd()).absolute()
images_dir = current_dir.joinpath('images')
images_dir.mkdir(parents=True, exist_ok = True)
```

[71]:
```
#Used article "Simple Image Classification with ResNet-50" from Medium
#I know I can loop this, but just having trouble getting it to work
#Load image1 and set target to 224, 224 since that is the format ResNet50 needs
img1 = image.load_img('images/bunny.jpg', target_size = (224, 224))
#show image
plt.imshow(img1)
```

[71]:
```
<matplotlib.image.AxesImage at 0x7fb08c25f640>
```



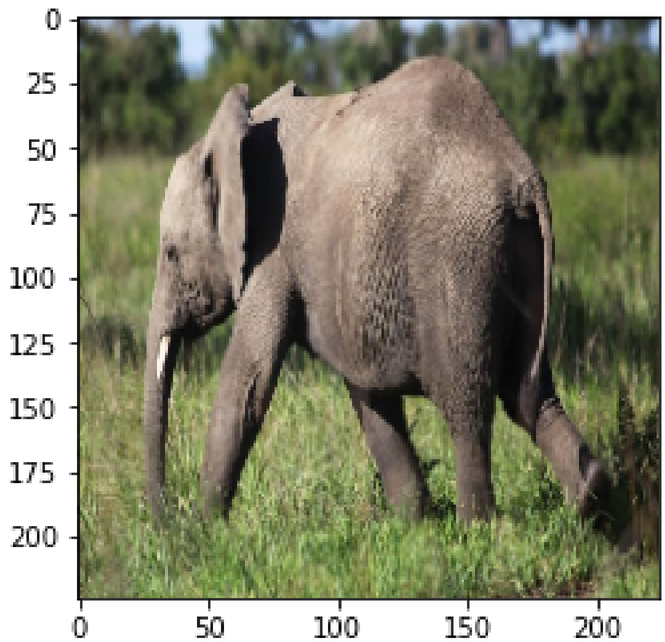
[72]:
```
#turn into a numpy array
img1 = image.img_to_array(img1)
#insert new axis that will appear at the axis position in expanded array shape
img1 = np.expand_dims(img1, axis = 0)
#Preprocessing the numpy array encoding a batch of images
img1 = preprocess_input(img1)
#instantiating the ResNet50 model
model = ResNet50(weights = 'imagenet')
#predicting on the model and printing the result
preds1 = model.predict(img1)
print('Predicted:', decode_predictions(preds1, top = 1)[0])
```

```
WARNING:tensorflow:5 out of the last 9 calls to <function
Model.make_predict_function.<locals>.predict_function at 0x7fb0206641f0>
triggered tf.function retracing. Tracing is expensive and the excessive number
of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2)
passing tensors with different shapes, (3) passing Python objects instead of
```

```
tensors. For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has experimental_relax_shapes=True option that relaxes argument
shapes that can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
Predicted: [('n02326432', 'hare', 0.5823781)]
```

[73]:
```
#Load image2 and set target to 224, 224 since that is the format ResNet50 needs
img2 = image.load_img('images/elephant.jpg', target_size = (224, 224))
#show image
plt.imshow(img2)
```

[73]:
```
<matplotlib.image.AxesImage at 0x7fb02052e5e0>
```

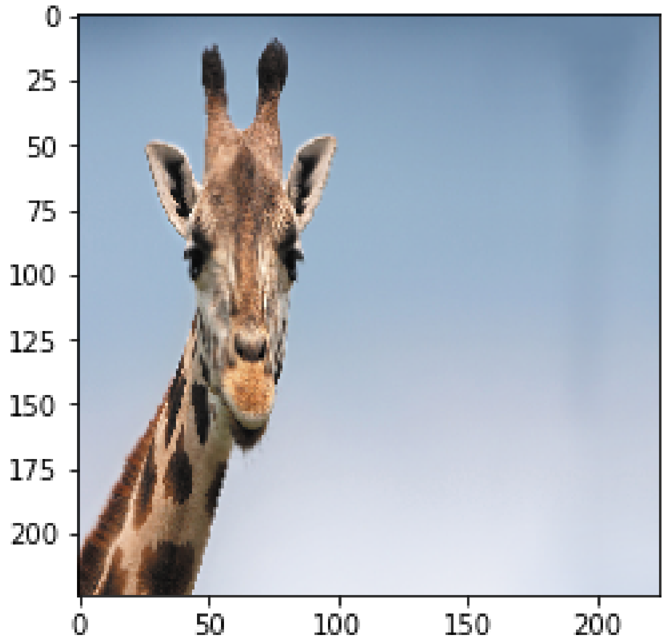


[74]:
```
#turn into a numpy array
img2 = image.img_to_array(img2)
#insert new axis that will appear at the axis position in expanded array shape
img2 = np.expand_dims(img2, axis = 0)
#Preprocessing the numpy array encoding a batch of images
img2 = preprocess_input(img2)
#instantiating the ResNet50 model
model2 = ResNet50(weights = 'imagenet')
#predicting on the model and printing the result
preds2 = model2.predict(img2)
print('Predicted:', decode_predictions(preds2, top = 1)[0])
```

```
Predicted: [('n01871265', 'tusker', 0.45374754)]
```

[75]:
```python
#Load image3 and set target to 224, 224 since that is the format ResNet50 needs
img3 = image.load_img('images/giraffe.png', target_size = (224, 224))
#show image
plt.imshow(img3)
```

[75]: <matplotlib.image.AxesImage at 0x7fb01855efd0>



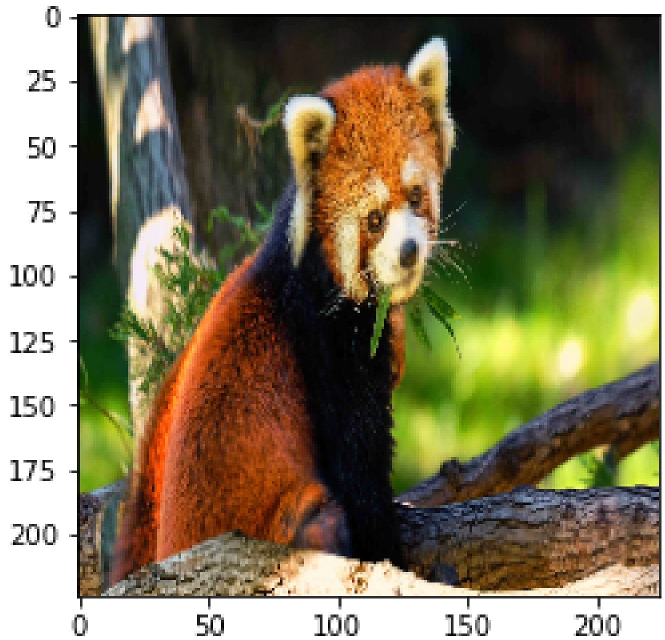
[76]:
```python
#turn into a numpy array
img3 = image.img_to_array(img3)
#insert new axis that will appear at the axis position in expanded array shape
img3 = np.expand_dims(img3, axis = 0)
#Preprocessing the numpy array encoding a batch of images
img3 = preprocess_input(img3)
#instantiating the ResNet50 model
model3 = ResNet50(weights = 'imagenet')
#predicting on the model and printing the result
preds3 = model.predict(img3)
print('Predicted:', decode_predictions(preds3, top = 1)[0])
```

```
Predicted: [('n02417914', 'ibex', 0.36325175)]
```

[77]:
```python
#Load image4 and set target to 224, 224 since that is the format ResNet50 needs
img4 = image.load_img('images/red_panda.png', target_size = (224, 224))
#show image
```

```
plt.imshow(img4)
```

[77]: <matplotlib.image.AxesImage at 0x7fb0182d3940>

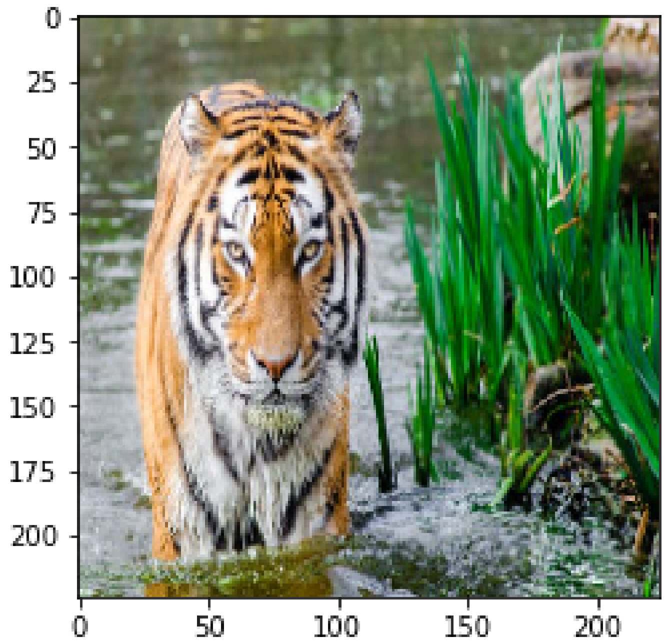


[78]:
```
#turn into a numpy array
img4 = image.img_to_array(img4)
#insert new axis that will appear at the axis position in expanded array shape
img4 = np.expand_dims(img4, axis = 0)
#Preprocessing the numpy array encoding a batch of images
img4 = preprocess_input(img4)
#instantiating the ResNet50 model
model4 = ResNet50(weights = 'imagenet')
#predicting on the model and printing the result
preds4 = model.predict(img4)
print('Predicted:', decode_predictions(preds4, top = 1)[0])
```

```
Predicted: [('n02509815', 'lesser_panda', 0.977285)]
```

[79]:
```
#Load image5 and set target to 224, 224 since that is the format ResNet50 needs
img5 = image.load_img('images/tiger.jpeg', target_size = (224, 224))
#show image
plt.imshow(img5)
```

[79]: <matplotlib.image.AxesImage at 0x7fb08c682e20>



[80]:

```
#turn into a numpy array
img5 = image.img_to_array(img5)
#insert new axis that will appear at the axis position in expanded array shape
img5 = np.expand_dims(img5, axis = 0)
#Preprocessing the numpy array encoding a batch of images
img5 = preprocess_input(img5)
#instantiating the ResNet50 model
model5 = ResNet50(weights = 'imagenet')
#predicting on the model and printing the result
preds5 = model.predict(img5)
print('Predicted:', decode_predictions(preds5, top = 1)[0])
```

```
Predicted: [('n02129604', 'tiger', 0.7966444)]
```