TFandKeras

February 19, 2023

[1]:
```
#DSC650
#Christine Orosco
```

0.1 Create create a ConvNet model that classifies images in the MNIST

[ ]:
```
# 6.1  Create create a ConvNet model that classifies images in the MNIST
#  digit dataset.
#  Save the model, predictions, metrics, and validation plots in
# the dsc650/assignments/assignment06/results directory
```

[2]:
```
from keras import layers
from keras import models
```

0.1.1 Build the model

[3]:
```
#  Create 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'))
```

[4]:
```
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
_________________________________________________________________
```

[5]:
```
# Add a classifier

model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

[6]:
```
# Display model results

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
_________________________________________________________________
```

0.1.2 Train the model

[7]:
```
# Train the model on the MNIST image dataset

from keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
```

[8]:
```
# assign dataset to variables
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# Reformat the data subsets

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)
```

[9]:
```
# Build amd fit the model
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.fit(train_images, train_labels, epochs=5, batch_size=64)
```

```
Epoch 1/5
938/938 [==============================] - 14s 15ms/step - loss: 0.1734 -
accuracy: 0.9456
Epoch 2/5
938/938 [==============================] - 13s 14ms/step - loss: 0.0474 -
accuracy: 0.9852
Epoch 3/5
938/938 [==============================] - 13s 14ms/step - loss: 0.0326 -
accuracy: 0.9902
Epoch 4/5
938/938 [==============================] - 13s 14ms/step - loss: 0.0244 -
accuracy: 0.9926
Epoch 5/5
938/938 [==============================] - 12s 13ms/step - loss: 0.0193 -
accuracy: 0.9937
```

[9]:
```
<tensorflow.python.keras.callbacks.History at 0x7fd2c077d730>
```

[10]:
```
# Eval the model on the test dataset

test_loss, test_acc = model.evaluate(test_images, test_labels)
```

```
313/313 [==============================] - 1s 4ms/step - loss: 0.0304 -
accuracy: 0.9908
```

0.2 Create a ConvNet model that classifies images CIFAR10 small images classification dataset

[ ]:
```
# Exercise 6.2
# Used the following sources:
# https://www.geeksforgeeks.org/cifar-10-image-classification-in-tensorflow/
# https://medium.com/@jayramchaudhury20/
  ↪project-on-image-classification-on-cifar-10-dataset-94db0ff6baf5
# https://keras.io/api/applications/#classify-imagenet-classes-with-resnet50
# https://www.tensorflow.org/tutorials/images/data_augmentation
# https://machinelearningmastery.com/
  ↪how-to-develop-a-cnn-from-scratch-for-cifar-10-photo-classification/
```

[11]:
```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.layers import Input, Conv2D, Dense, Flatten, Dropout
from tensorflow.keras.layers import GlobalMaxPooling2D, MaxPooling2D
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.models import Model
```

[12]:
```
# Load in the data
cifar10 = tf.keras.datasets.cifar10

# Distribute data to train and test set
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
print(x_train.shape, y_train.shape, x_test.shape, y_test.shape)
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 47s 0us/step
(50000, 32, 32, 3) (50000, 1) (10000, 32, 32, 3) (10000, 1)
```

[13]:
```
# Reduce pixel values
x_train, x_test = x_train / 255.0, x_test / 255.0

# flatten the label values
y_train, y_test = y_train.flatten(), y_test.flatten()
```

[ ]:
```
### Build the model
```

[15]:
```
# Build the model
# Use without the dropout

K = len(set(y_train))

# calculate total number of classes
# for output layer
print("number of classes:", K)


# Build the model using the functional API
# input layer
i = Input(shape=x_train[0].shape)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(i)
x = BatchNormalization()(x)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2))(x)

x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2))(x)

x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2))(x)

x = Flatten()(x)

# Hidden layer
x = Dense(1024, activation='relu')(x)


# last hidden layer i.e.. output layer
x = Dense(K, activation='softmax')(x)

model = Model(i, x)

# model description
model.summary()
```

```
number of classes: 10
Model: "model"
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 32, 32, 3)]       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 32, 32, 32)        896
_________________________________________________________________
batch_normalization (BatchNo (None, 32, 32, 32)        128
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 32, 32, 32)        9248
_________________________________________________________________
batch_normalization_1 (Batch (None, 32, 32, 32)        128
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 16, 16, 64)        18496
_________________________________________________________________
batch_normalization_2 (Batch (None, 16, 16, 64)        256
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 16, 16, 64)        36928
_________________________________________________________________
batch_normalization_3 (Batch (None, 16, 16, 64)        256
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 8, 8, 128)         73856
_________________________________________________________________
batch_normalization_4 (Batch (None, 8, 8, 128)         512
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 8, 8, 128)         147584
_________________________________________________________________
batch_normalization_5 (Batch (None, 8, 8, 128)         512
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_2 (Dense)              (None, 1024)              2098176
_________________________________________________________________
dense_3 (Dense)              (None, 10)                10250
=================================================================
Total params: 2,397,226
Trainable params: 2,396,330
Non-trainable params: 896
_________________________________________________________________
```

[16]:
```
# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

0.2.1 Train the model

[17]:
```
# Train the model

new_mod = model.fit(x_train, y_train, validation_data=(x_test, y_test),␣
 ↪epochs=20)
```

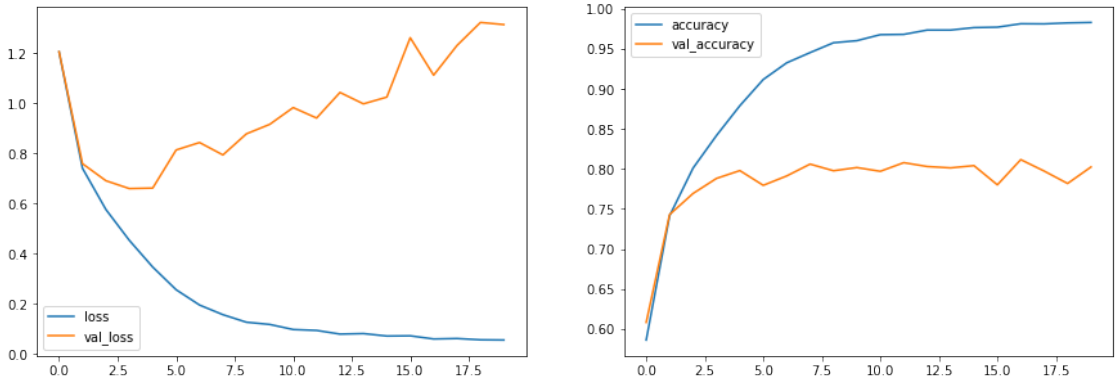
```
Epoch 1/20
1563/1563 [==============================] - 87s 56ms/step - loss: 1.2052 -
accuracy: 0.5864 - val_loss: 1.2062 - val_accuracy: 0.6084
Epoch 2/20
1563/1563 [==============================] - 87s 56ms/step - loss: 0.7407 -
accuracy: 0.7415 - val_loss: 0.7591 - val_accuracy: 0.7430
Epoch 3/20
1563/1563 [==============================] - 87s 56ms/step - loss: 0.5762 -
accuracy: 0.8009 - val_loss: 0.6911 - val_accuracy: 0.7693
Epoch 4/20
1563/1563 [==============================] - 88s 56ms/step - loss: 0.4531 -
accuracy: 0.8416 - val_loss: 0.6597 - val_accuracy: 0.7881
Epoch 5/20
1563/1563 [==============================] - 86s 55ms/step - loss: 0.3465 -
accuracy: 0.8789 - val_loss: 0.6617 - val_accuracy: 0.7979
Epoch 6/20
1563/1563 [==============================] - 87s 55ms/step - loss: 0.2559 -
accuracy: 0.9116 - val_loss: 0.8137 - val_accuracy: 0.7795
Epoch 7/20
1563/1563 [==============================] - 86s 55ms/step - loss: 0.1947 -
accuracy: 0.9324 - val_loss: 0.8436 - val_accuracy: 0.7912
Epoch 8/20
1563/1563 [==============================] - 87s 56ms/step - loss: 0.1562 -
accuracy: 0.9452 - val_loss: 0.7940 - val_accuracy: 0.8060
Epoch 9/20
1563/1563 [==============================] - 88s 56ms/step - loss: 0.1262 -
accuracy: 0.9575 - val_loss: 0.8781 - val_accuracy: 0.7977
Epoch 10/20
1563/1563 [==============================] - 87s 56ms/step - loss: 0.1172 -
accuracy: 0.9601 - val_loss: 0.9162 - val_accuracy: 0.8017
Epoch 11/20
1563/1563 [==============================] - 87s 56ms/step - loss: 0.0970 -
accuracy: 0.9676 - val_loss: 0.9830 - val_accuracy: 0.7970
Epoch 12/20
1563/1563 [==============================] - 87s 56ms/step - loss: 0.0933 -
```

```
accuracy: 0.9680 - val_loss: 0.9413 - val_accuracy: 0.8078
Epoch 13/20
1563/1563 [==============================] - 87s 56ms/step - loss: 0.0787 -
accuracy: 0.9734 - val_loss: 1.0435 - val_accuracy: 0.8030
Epoch 14/20
1563/1563 [==============================] - 87s 55ms/step - loss: 0.0808 -
accuracy: 0.9734 - val_loss: 0.9979 - val_accuracy: 0.8013
Epoch 15/20
1563/1563 [==============================] - 87s 56ms/step - loss: 0.0709 -
accuracy: 0.9765 - val_loss: 1.0248 - val_accuracy: 0.8041
Epoch 16/20
1563/1563 [==============================] - 87s 56ms/step - loss: 0.0718 -
accuracy: 0.9770 - val_loss: 1.2619 - val_accuracy: 0.7801
Epoch 17/20
1563/1563 [==============================] - 87s 56ms/step - loss: 0.0593 -
accuracy: 0.9813 - val_loss: 1.1124 - val_accuracy: 0.8116
Epoch 18/20
1563/1563 [==============================] - 87s 56ms/step - loss: 0.0611 -
accuracy: 0.9811 - val_loss: 1.2313 - val_accuracy: 0.7973
Epoch 19/20
1563/1563 [==============================] - 86s 55ms/step - loss: 0.0558 -
accuracy: 0.9823 - val_loss: 1.3230 - val_accuracy: 0.7818
Epoch 20/20
1563/1563 [==============================] - 87s 56ms/step - loss: 0.0550 -
accuracy: 0.9829 - val_loss: 1.3144 - val_accuracy: 0.8024
```

0.2.2 Plot the Learning Curves

[19]:
```
# Plot Learning Curves

import pandas as pd

fig, ax = plt.subplots(1,2, figsize=(15,5))

history_df = pd.DataFrame(new_mod.history)
history_df[['loss', 'val_loss']].plot(kind='line', ax=ax[0])
history_df[['accuracy', 'val_accuracy']].plot(kind='line', ax=ax[1])
```

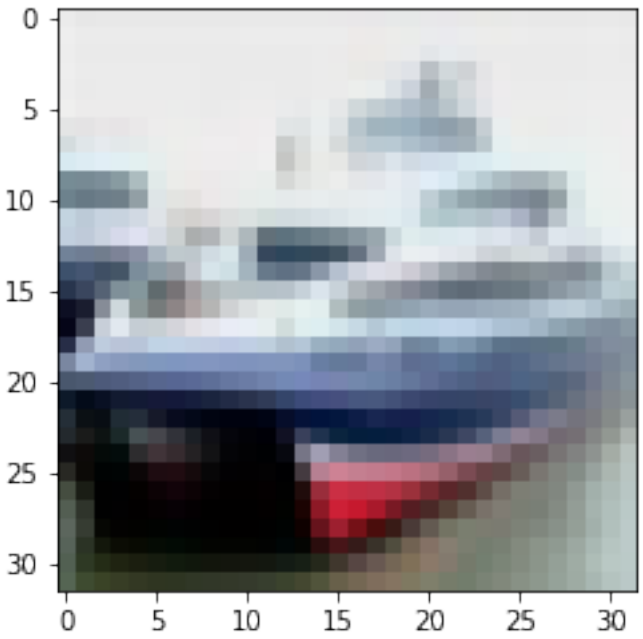
[19]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd2903e3310>
```



0.2.3 Generate Predictions

[23]:
```
# Generate a Prediction

labels = '''airplane automobile bird cat deer dog frog horse ship truck'''.
  ↪split()

# select the image from our test dataset
image_number = 1

# display the image
plt.imshow(x_test[image_number])

# load the image in an array
n = np.array(x_test[image_number])

# reshape the array
p = n.reshape(1, 32, 32, 3)

# pass in the network for prediction and
# save the predicted label
predicted_label = labels[model.predict(p).argmax()]

# load the original label
original_label = labels[y_test[image_number]]

# display the result
print(f"Original label is {original_label} and predicted label is␣
  ↪{predicted_label}")
```

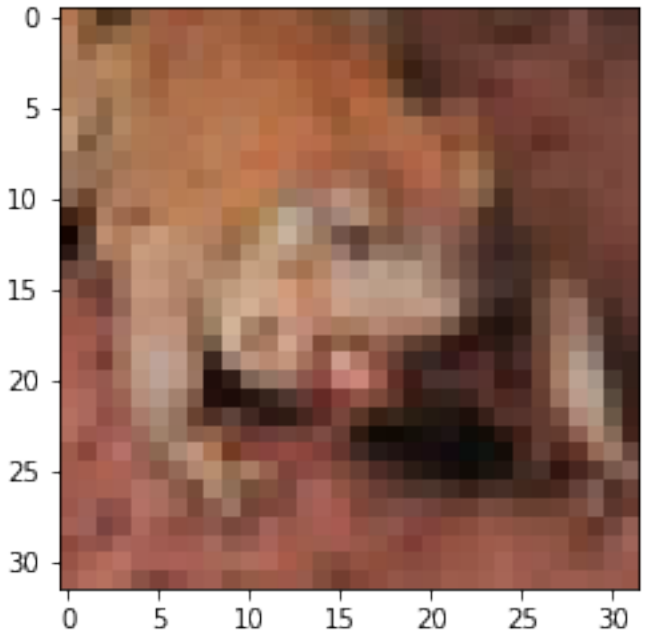
```
Original label is ship and predicted label is automobile
```



[24]:
```
# Generate a Prediction

labels = '''airplane automobile bird cat deer dog frog horse ship truck'''.
 ↪split()

# select the image from our test dataset
image_number = 5

# display the image
plt.imshow(x_test[image_number])

# load the image in an array
n = np.array(x_test[image_number])

# reshape the array
p = n.reshape(1, 32, 32, 3)

# pass in the network for prediction and
# save the predicted label
predicted_label = labels[model.predict(p).argmax()]

# load the original label
original_label = labels[y_test[image_number]]

# display the result
```

```
print(f"Original label is {original_label} and predicted label is␣
 ↪{predicted_label}")
```

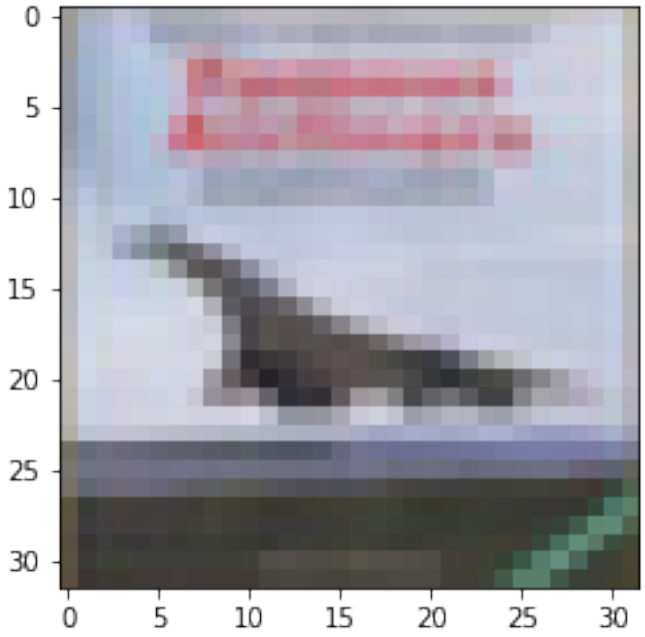
```
Original label is frog and predicted label is frog
```



[25]:
```
# Generate a Prediction

labels = '''airplane automobile bird cat deer dog frog horse ship truck'''.
 ↪split()

# select the image from our test dataset
image_number = 3

# display the image
plt.imshow(x_test[image_number])

# load the image in an array
n = np.array(x_test[image_number])

# reshape the array
p = n.reshape(1, 32, 32, 3)

# pass in the network for prediction and
# save the predicted label
predicted_label = labels[model.predict(p).argmax()]
```

```
# load the original label
original_label = labels[y_test[image_number]]

# display the result
print(f"Original label is {original_label} and predicted label is␣
 ↪{predicted_label}")
```

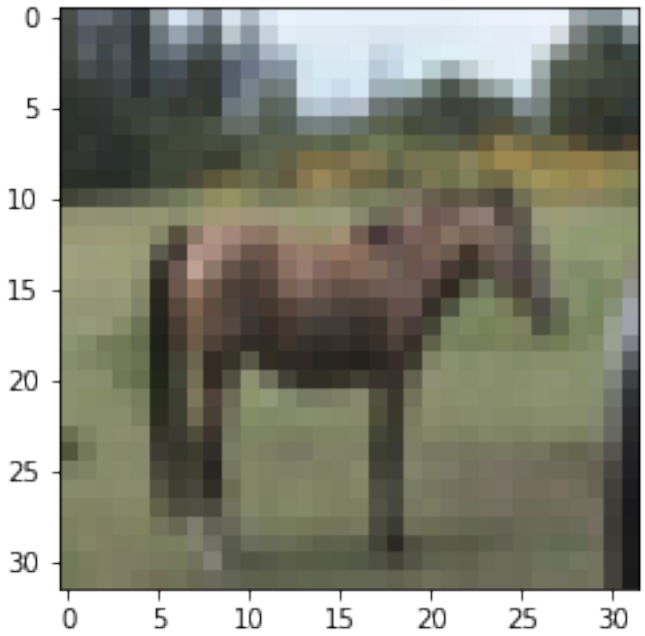
```
Original label is airplane and predicted label is airplane
```



[26]:
```
# Generate a Prediction

labels = '''airplane automobile bird cat deer dog frog horse ship truck'''.
 ↪split()

# select the image from our test dataset
image_number = 9999

# display the image
plt.imshow(x_test[image_number])

# load the image in an array
n = np.array(x_test[image_number])

# reshape the array
p = n.reshape(1, 32, 32, 3)
```

```
# pass in the network for prediction and
# save the predicted label
predicted_label = labels[model.predict(p).argmax()]

# load the original label
original_label = labels[y_test[image_number]]

# display the result
print(f"Original label is {original_label} and predicted label is␣
 ↪{predicted_label}")
```

```
Original label is horse and predicted label is horse
```



[27]:
```
# 6.2b Build the model with dropout

K = len(set(y_train))

# calculate total number of classes
# for output layer
print("number of classes:", K)

# Build the model using the functional API
# input layer
i = Input(shape=x_train[0].shape)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(i)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
```

```
x = MaxPooling2D((2, 2))(x)

x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = MaxPooling2D((2, 2))(x)

x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
x = MaxPooling2D((2, 2))(x)

x = Flatten()(x)
x = Dropout(0.25)(x)

# Hidden layer
x = Dense(1024, activation='relu')(x)
x = Dropout(0.5)(x)

# last hidden layer i.e.. output layer
x = Dense(K, activation='softmax')(x)

model = Model(i, x)

# model description
model.summary()
```

```
number of classes: 10
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 32, 32, 3)]       0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 32, 32, 32)        896
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 32, 32, 32)        9248
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 16, 16, 64)        18496
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 8, 8, 128)         73856
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 2048)              0
_________________________________________________________________
dropout (Dropout)            (None, 2048)              0
```

```
_________________________________________________________________
dense_4 (Dense)              (None, 1024)              2098176
_________________________________________________________________
dropout_1 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_5 (Dense)              (None, 10)                10250
=================================================================
Total params: 2,210,922
Trainable params: 2,210,922
Non-trainable params: 0
_________________________________________________________________
```

[28]:
```
# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

[29]:
```
# 6.2b Fit with data augmentation

batch_size = 32
data_generator = tf.keras.preprocessing.image.ImageDataGenerator(
width_shift_range=0.1, height_shift_range=0.1, horizontal_flip=True)

train_generator = data_generator.flow(x_train, y_train, batch_size)
steps_per_epoch = x_train.shape[0] // batch_size

new_model = model.fit(train_generator, validation_data=(x_test, y_test),
                      steps_per_epoch=steps_per_epoch, epochs=20)
```

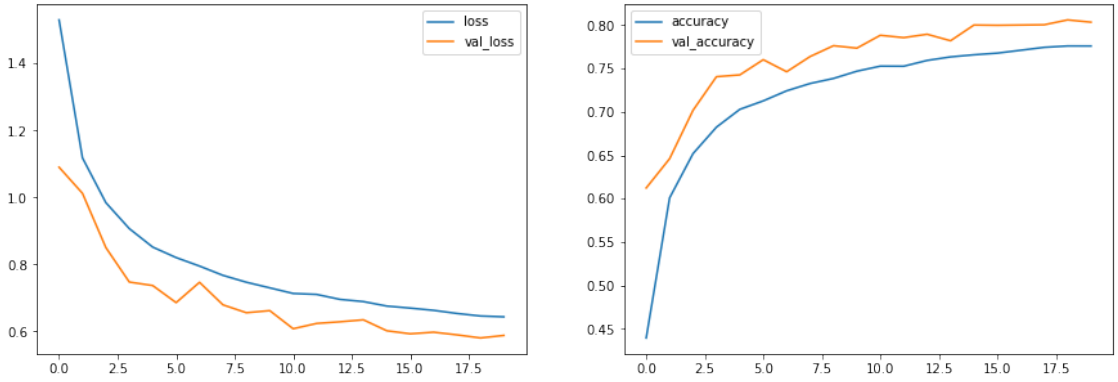
```
Epoch 1/20
1562/1562 [==============================] - 51s 33ms/step - loss: 1.5304 -
accuracy: 0.4397 - val_loss: 1.0905 - val_accuracy: 0.6124
Epoch 2/20
1562/1562 [==============================] - 51s 33ms/step - loss: 1.1183 -
accuracy: 0.6010 - val_loss: 1.0125 - val_accuracy: 0.6461
Epoch 3/20
1562/1562 [==============================] - 51s 33ms/step - loss: 0.9846 -
accuracy: 0.6519 - val_loss: 0.8504 - val_accuracy: 0.7018
Epoch 4/20
1562/1562 [==============================] - 51s 33ms/step - loss: 0.9071 -
accuracy: 0.6824 - val_loss: 0.7476 - val_accuracy: 0.7405
Epoch 5/20
1562/1562 [==============================] - 51s 33ms/step - loss: 0.8519 -
accuracy: 0.7029 - val_loss: 0.7371 - val_accuracy: 0.7426
Epoch 6/20
1562/1562 [==============================] - 51s 33ms/step - loss: 0.8206 -
accuracy: 0.7127 - val_loss: 0.6861 - val_accuracy: 0.7601
Epoch 7/20
```

```
1562/1562 [==============================] - 51s 33ms/step - loss: 0.7952 -
accuracy: 0.7243 - val_loss: 0.7468 - val_accuracy: 0.7462
Epoch 8/20
1562/1562 [==============================] - 52s 33ms/step - loss: 0.7676 -
accuracy: 0.7327 - val_loss: 0.6795 - val_accuracy: 0.7637
Epoch 9/20
1562/1562 [==============================] - 51s 33ms/step - loss: 0.7470 -
accuracy: 0.7385 - val_loss: 0.6559 - val_accuracy: 0.7762
Epoch 10/20
1562/1562 [==============================] - 52s 33ms/step - loss: 0.7302 -
accuracy: 0.7468 - val_loss: 0.6622 - val_accuracy: 0.7734
Epoch 11/20
1562/1562 [==============================] - 52s 33ms/step - loss: 0.7134 -
accuracy: 0.7527 - val_loss: 0.6081 - val_accuracy: 0.7882
Epoch 12/20
1562/1562 [==============================] - 52s 33ms/step - loss: 0.7106 -
accuracy: 0.7526 - val_loss: 0.6239 - val_accuracy: 0.7855
Epoch 13/20
1562/1562 [==============================] - 52s 33ms/step - loss: 0.6958 -
accuracy: 0.7592 - val_loss: 0.6289 - val_accuracy: 0.7894
Epoch 14/20
1562/1562 [==============================] - 51s 33ms/step - loss: 0.6892 -
accuracy: 0.7633 - val_loss: 0.6347 - val_accuracy: 0.7819
Epoch 15/20
1562/1562 [==============================] - 51s 33ms/step - loss: 0.6758 -
accuracy: 0.7658 - val_loss: 0.6020 - val_accuracy: 0.8000
Epoch 16/20
1562/1562 [==============================] - 51s 33ms/step - loss: 0.6698 -
accuracy: 0.7677 - val_loss: 0.5930 - val_accuracy: 0.7997
Epoch 17/20
1562/1562 [==============================] - 51s 33ms/step - loss: 0.6630 -
accuracy: 0.7710 - val_loss: 0.5978 - val_accuracy: 0.8000
Epoch 18/20
1562/1562 [==============================] - 51s 33ms/step - loss: 0.6537 -
accuracy: 0.7743 - val_loss: 0.5898 - val_accuracy: 0.8003
Epoch 19/20
1562/1562 [==============================] - 51s 33ms/step - loss: 0.6462 -
accuracy: 0.7759 - val_loss: 0.5806 - val_accuracy: 0.8059
Epoch 20/20
1562/1562 [==============================] - 51s 33ms/step - loss: 0.6437 -
accuracy: 0.7759 - val_loss: 0.5881 - val_accuracy: 0.8033
```

[30]:
```
# Plot Learning Curves

fig, ax = plt.subplots(1,2, figsize=(15,5))

history_df = pd.DataFrame(new_model.history)
```

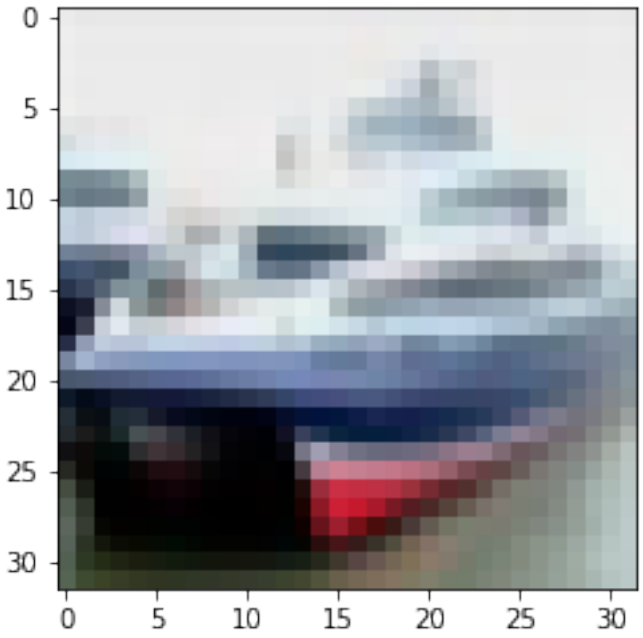
```
history_df[['loss', 'val_loss']].plot(kind='line', ax=ax[0])
history_df[['accuracy', 'val_accuracy']].plot(kind='line', ax=ax[1])
```

[30]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd2904d4100>



[31]:
```
# Generate a Prediction

labels = '''airplane automobile bird cat deer dog frog horse ship truck'''.
  ↪split()

# select the image from the test dataset
image_number = 1

# display the image
plt.imshow(x_test[image_number])

# load the image in an array
n = np.array(x_test[image_number])

# reshape the array
p = n.reshape(1, 32, 32, 3)

# pass in the network for prediction and
# save the predicted label
predicted_label = labels[model.predict(p).argmax()]

# load the original label
original_label = labels[y_test[image_number]]

# display the result
print(f"Original label is {original_label} and predicted label is␣
  ↪{predicted_label}")
```

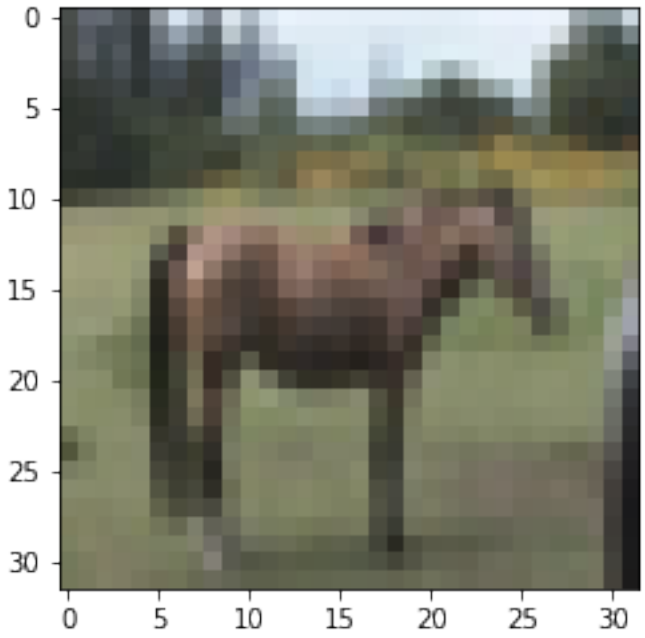
```
Original label is ship and predicted label is ship
```



[32]:
```
# Generate a Prediction

labels = '''airplane automobile bird cat deer dog frog horse ship truck'''.
 ↪split()

# select the image from the test dataset
image_number = 9999

# display the image
plt.imshow(x_test[image_number])

# load the image in an array
n = np.array(x_test[image_number])

# reshape the array
p = n.reshape(1, 32, 32, 3)

# pass in the network for prediction and
# save the predicted label
predicted_label = labels[model.predict(p).argmax()]

# load the original label
original_label = labels[y_test[image_number]]
```

```
# display the result
print(f"Original label is {original_label} and predicted label is␣
 ↪{predicted_label}")
```

```
Original label is horse and predicted label is horse
```



0.3 Load the ResNet50 model

[5]:
```
#6.3 ResNET50 model

from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.resnet50 import preprocess_input,␣
 ↪decode_predictions
import numpy as np
import PIL
```

[2]:
```
model = ResNet50(weights='imagenet')

img_path = 'precious_stryder.jpeg'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)
```

```
preds = model.predict(x)
# decode the results into a list of tuples (class, description, probability)
# (one such list for each sample in the batch)
print('Predicted:', decode_predictions(preds, top=3)[0])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels.h5
102973440/102967424 [==============================] - 17s 0us/step
Downloading data from https://storage.googleapis.com/download.tensorflow.org/dat
a/imagenet_class_index.json
40960/35363 [==================================] - 0s 1us/step
Predicted: [('n02102318', 'cocker_spaniel', 0.58005065), ('n02086240', 'Shih-
Tzu', 0.16435128), ('n02100735', 'English_setter', 0.06059162)]
```

[3]:
```
# Display my Shih Tzu fur babies
from IPython import display
display.Image("precious_stryder.jpeg")
```

[3]:

[6]:
```
model = ResNet50(weights='imagenet')

img_path = 'rosesv1.jpeg'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

preds = model.predict(x)
# decode the results into a list of tuples (class, description, probability)
```

```
# (one such list for each sample in the batch)
print('Predicted:', decode_predictions(preds, top=3)[0])
```

```
Downloading data from https://storage.googleapis.com/download.tensorflow.org/dat
a/imagenet_class_index.json
40960/35363 [==================================] - 0s 1us/step
Predicted: [('n03991062', 'pot', 0.87892425), ('n03930313', 'picket_fence',
0.040227942), ('n02797295', 'barrow', 0.01682531)]
```

[7]:
```
# Display rose bush
from IPython import display
display.Image("rosesv1.jpeg")
```

[7]:

[8]:
```
img_path = 'aidyn.jpeg'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

preds = model.predict(x)
# decode the results into a list of tuples (class, description, probability)
# (one such list for each sample in the batch)
print('Predicted:', decode_predictions(preds, top=3)[0])
```

```
Predicted: [('n04162706', 'seat_belt', 0.18271334), ('n03938244', 'pillow',
0.18187921), ('n02786058', 'Band_Aid', 0.14660415)]
```

[9]:
```
# Display my sweetie pie
from IPython import display
display.Image("aidyn.jpeg")
```

[9]:

[ ]: