Assignment_9_1

Technological Institute of the Philippines	Quezon City - Computer Engineering
Course Code:	CPE 019
Code Title:	Emerging Technologies in CpE 2
Summer	AY 2024 - 2025

Assignment 9.1	**Convolutional Neural Network**
Name	Calvadores, Kelly Joseph
Section	CPE32S1
Date Performed:	July 07, 2024
Date Submitted:	July 09, 2024
Instructor:	Engr. Roman M. Richard

Choose any dataset applicable to an image classification problem¶

Explain your datasets and the problem being addressed.¶

- In this activty, I chose 2 different dataset that i had used, the first is the cifar10 and the second is fashion_mnist. The problem the is being addressed is to classify both dataset into one of the 10 items in their own category.

Show evidence that you can do the following:¶

Using your dataset, create a baseline model of the CNN¶

In [ ]:

```
import time
Start_Time = time.time()
from numpy import mean
from numpy import std
from matplotlib import pyplot as plt
from sklearn.model_selection import KFold
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten
from tensorflow.keras.optimizers import SGD
```

In [ ]:

```
def LoadDataset():
  (trainX, trainY), (testX, testY) = cifar10.load_data()
  trainX = trainX.reshape((trainX.shape[0], 32, 32, 3))
  testX = testX.reshape((testX.shape[0], 32, 32, 3))

  trainY = to_categorical(trainY)
  testY = to_categorical(testY)
  return trainX, trainY, testX, testY

def prep_pixels(train, test):
    # convert from integers to floats
    train_norm = train.astype('float32')
    test_norm = test.astype('float32')
    # normalize to range 0-1
    train_norm = train_norm / 255.0
    test_norm = test_norm / 255.0
    # return normalized images
    return train_norm, test_norm
```

In [ ]:

```
def DefineModel():
  model = Sequential()
  model.add(Conv2D(32, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same', input_shape=(32, 32, 3)))
  model.add(MaxPooling2D((2, 2)))
  model.add(Flatten())
  model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
  model.add(Dense(10, activation='softmax'))
  opt = SGD(lr=0.001, momentum=0.9)
  model.compile(optimizer=opt, loss='categorical_crossentropy',
metrics=['accuracy'])
  return model

def EvaluateModel(dataX, dataY, n_folds=5):
  scores, histories = list(), list()
  kfold = KFold(n_folds, shuffle=True, random_state=1)
  for train_ix, test_ix in kfold.split(dataX):
    model = DefineModel()
    trainX, trainy, testX, testy = dataX[train_ix], dataY[train_ix],
dataX[test_ix], dataY[test_ix]
    history = model.fit(trainX, trainy, epochs=100, batch_size=1000,
```

```
validation_data=(testX, testy), verbose=0)
    _, acc = model.evaluate(testX, testy, verbose=0)
    print('> %.3f' % (acc * 100.0))
    scores.append(acc)
    histories.append(history)
  return scores, histories
```

In [ ]:

```
def SummarizeHistory(histories):
  for i in range(len(histories)):
    plt.subplot(2, 1, 1)
    plt.title('Cross Entropy Loss')
    plt.plot(histories[i].history['loss'], color='blue', label='train')
    plt.plot(histories[i].history['val_loss'], color='orange', label='test')
    plt.subplot(2, 1, 2)
    plt.title('Classification Accuracy')
    plt.plot(histories[i].history['accuracy'], color='blue', label='train')
    plt.plot(histories[i].history['val_accuracy'], color='orange',
label='test')
  plt.show()

def SummarizePerformance(scores):
  print('Accuracy: mean=%.3f std=%.3f, n=%d' % (mean(scores)*100,
std(scores)*100, len(scores)))
  plt.boxplot(scores)
  plt.show()
```

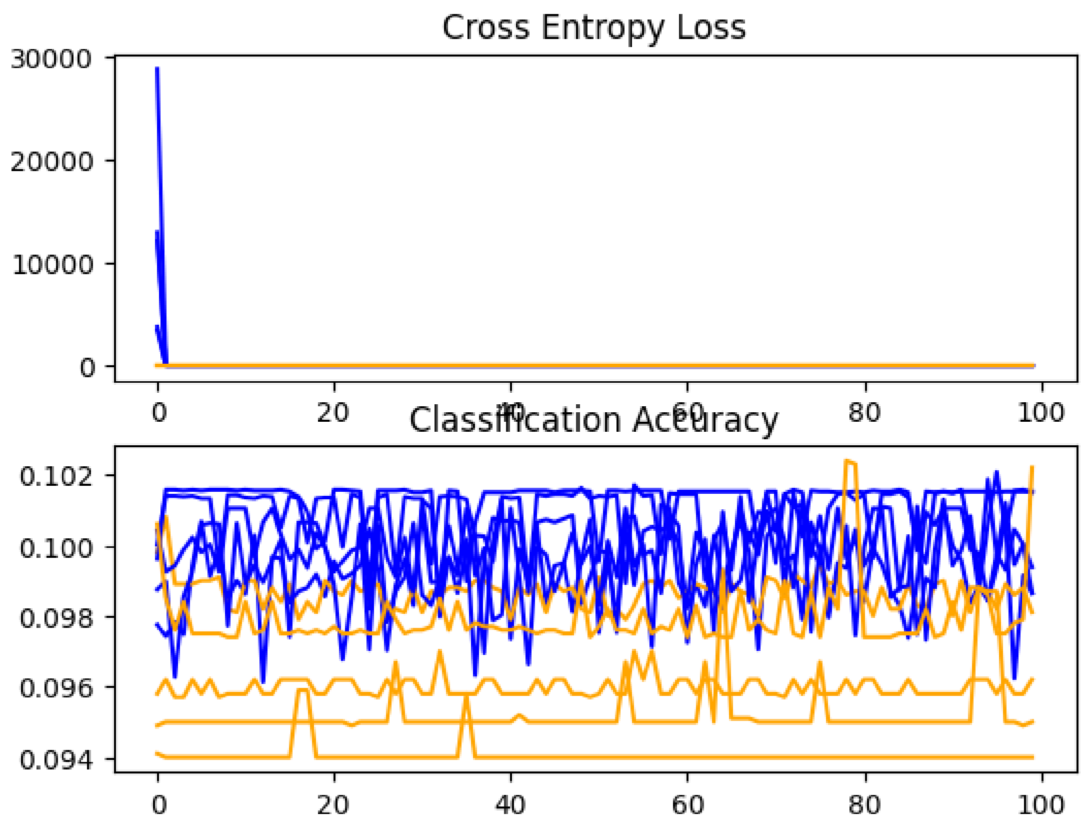
In [ ]:

```
def RunTestHarness():
  trainX, trainY, testX, testY = LoadDataset()
  scores, histories = EvaluateModel(trainX, trainY)
  SummarizeHistory(histories)
  SummarizePerformance(scores)

RunTestHarness()

End_Time = time.time()
Elapsed_Time = End_Time - Start_Time
print("Elapsed Time:", Elapsed_Time, "seconds")
```

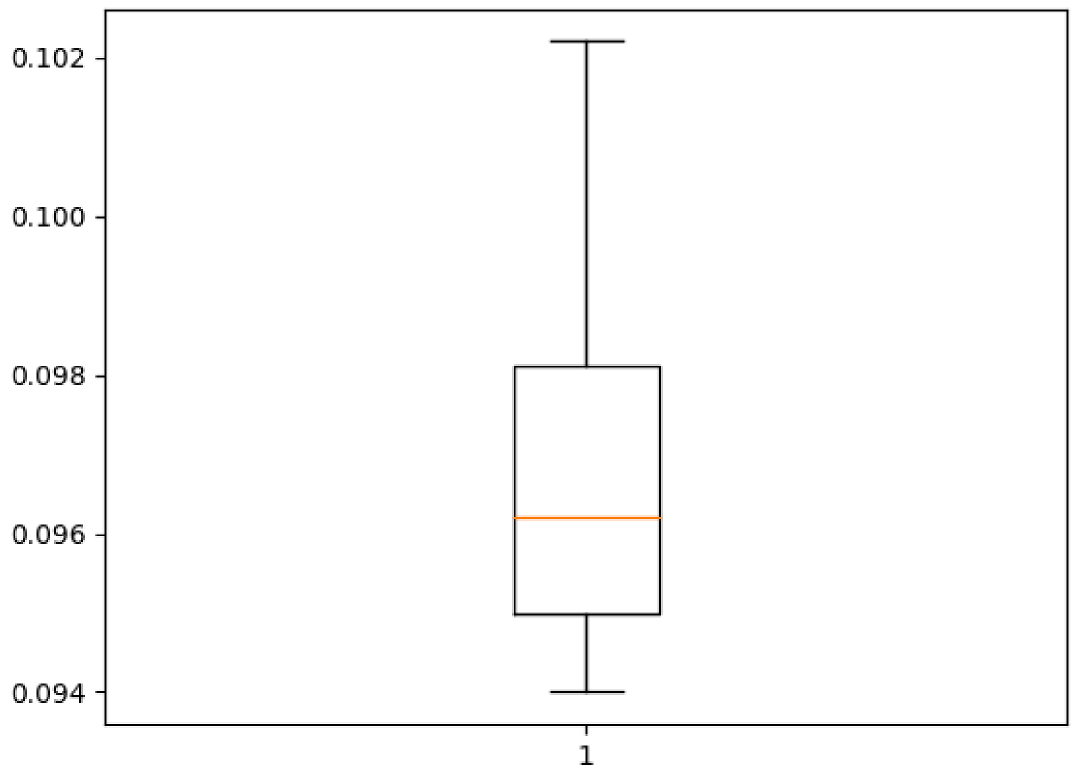
```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use
`learning_rate` or use the legacy optimizer,
e.g.,tf.keras.optimizers.legacy.SGD.
WARNING:absl:`lr` is deprecated in Keras optimizer, please use
`learning_rate` or use the legacy optimizer,
e.g.,tf.keras.optimizers.legacy.SGD.
```

```
> 9.810
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use
`learning_rate` or use the legacy optimizer,
e.g.,tf.keras.optimizers.legacy.SGD.

> 9.620

WARNING:absl:`lr` is deprecated in Keras optimizer, please use
`learning_rate` or use the legacy optimizer,
e.g.,tf.keras.optimizers.legacy.SGD.

> 9.500

WARNING:absl:`lr` is deprecated in Keras optimizer, please use
`learning_rate` or use the legacy optimizer,
e.g.,tf.keras.optimizers.legacy.SGD.

> 9.400
> 10.220
```




```
Accuracy: mean=9.710 std=0.289, n=5
```




```
Elapsed Time: 1963.3227407932281 seconds
```

Observation: As seen in the figures or images, the model is messy compare in the modules, the reason is that it might learn too well and create a noise instead of going along the line.

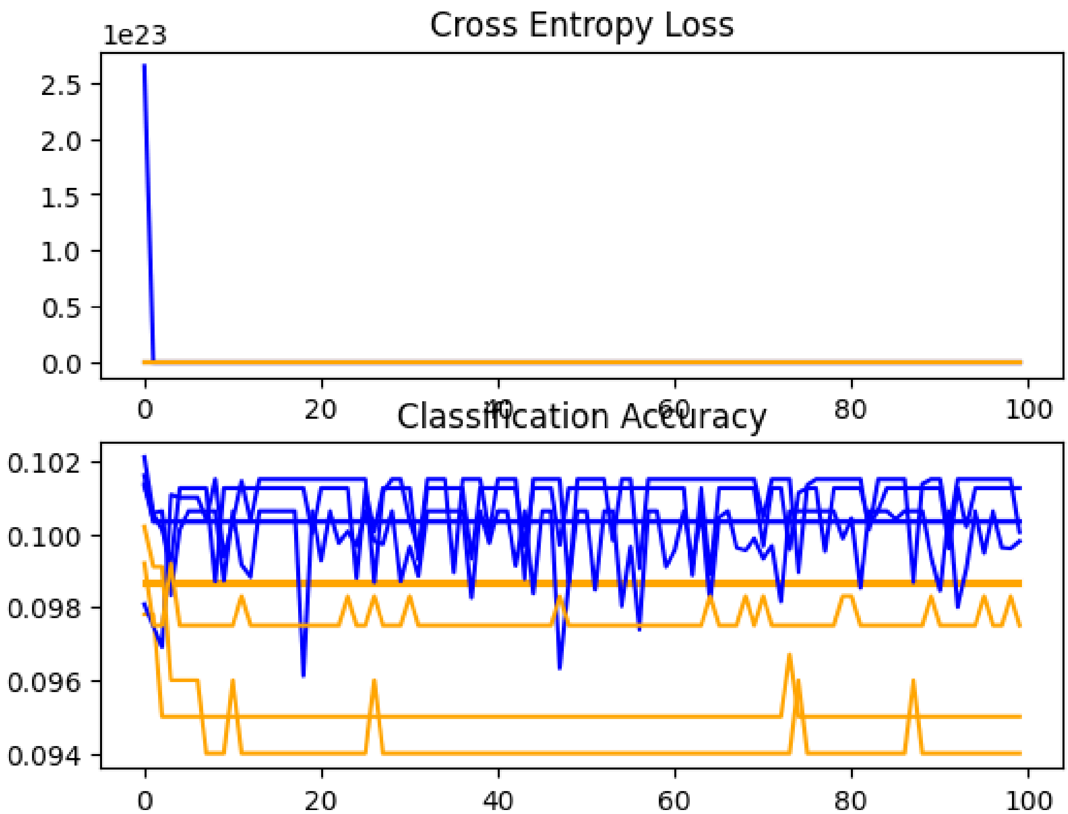
In [ ]:

```
def DefineModel():
  model = Sequential()
  model.add(Conv2D(32, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same', input_shape=(32, 32, 3)))
  model.add(MaxPooling2D((2, 2)))
  model.add(Conv2D(64, (3, 3), activation='relu',
kernel_initializer='he_uniform'))
  model.add(MaxPooling2D((2, 2)))
  model.add(Flatten())
  model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
  model.add(Dense(10, activation='softmax'))
  opt = SGD(lr=0.001, momentum=0.9)
  model.compile(optimizer=opt, loss='categorical_crossentropy',
metrics=['accuracy'])
  return model
```

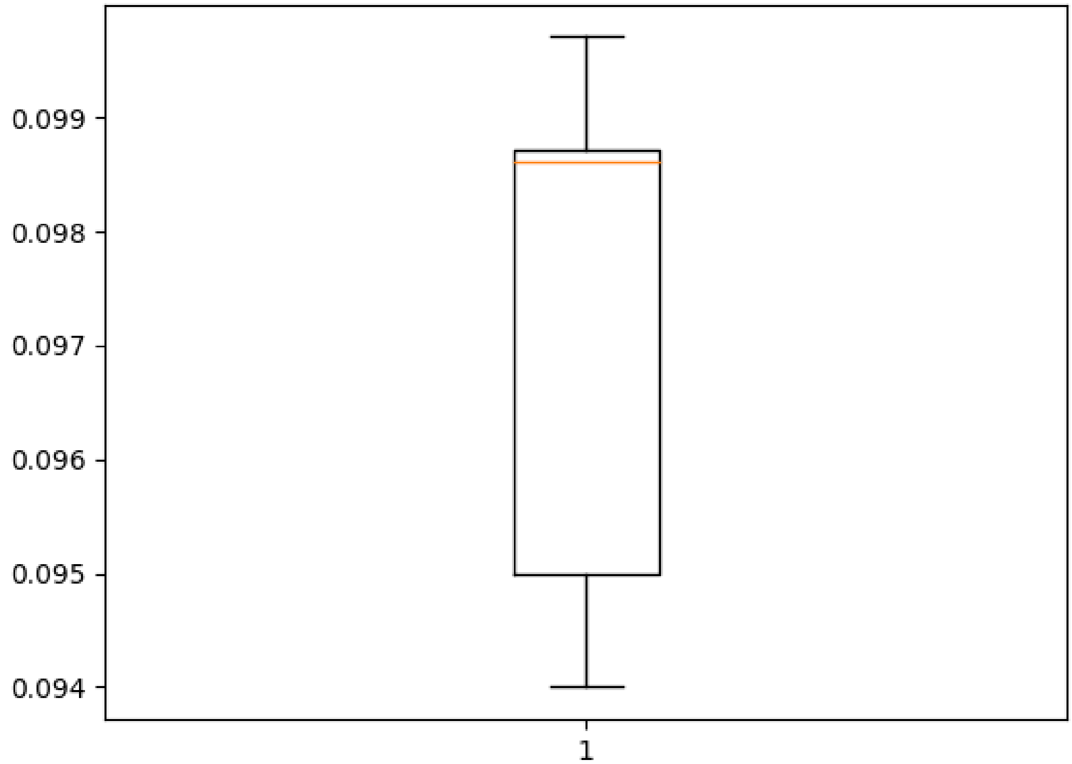
In [ ]:

```
RunTestHarness()

End_Time = time.time()
Elapsed_Time = End_Time - Start_Time
print("Elapsed Time:", Elapsed_Time, "seconds")
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use
`learning_rate` or use the legacy optimizer,
e.g.,tf.keras.optimizers.legacy.SGD.
WARNING:absl:`lr` is deprecated in Keras optimizer, please use
`learning_rate` or use the legacy optimizer,
e.g.,tf.keras.optimizers.legacy.SGD.

> 9.860

WARNING:absl:`lr` is deprecated in Keras optimizer, please use
`learning_rate` or use the legacy optimizer,
e.g.,tf.keras.optimizers.legacy.SGD.

> 9.870

WARNING:absl:`lr` is deprecated in Keras optimizer, please use
`learning_rate` or use the legacy optimizer,
e.g.,tf.keras.optimizers.legacy.SGD.

> 9.500

WARNING:absl:`lr` is deprecated in Keras optimizer, please use
`learning_rate` or use the legacy optimizer,
e.g.,tf.keras.optimizers.legacy.SGD.

> 9.400
> 9.970
```




```
Accuracy: mean=9.720 std=0.226, n=5
```




```
Elapsed Time: 2590.1185982227325 seconds
```

Observation: As seen in the images above, it is much clear compare to the previous but not quite enough.

In [ ]:

```
def RunTestHarness():
  trainX, trainY, testX, testY = LoadDataset()
  trainX, testX = prep_pixels(trainX, testX)
  model = DefineModel()
  model.fit(trainX, trainY, epochs=100, batch_size=1000, verbose=0)
  model.save('/content/drive/MyDrive/CPE 019 (Retake)/Assignment 
9.1/CNNmodel.h5')

RunTestHarness()

End_Time = time.time()
Elapsed_Time = End_Time - Start_Time
print("Elapsed Time:", Elapsed_Time, "seconds")
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use 
`learning_rate` or use the legacy optimizer, 
e.g.,tf.keras.optimizers.legacy.SGD.

Elapsed Time: 2773.8991510868073 seconds
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103:
UserWarning: You are saving your model as an HDF5 file via `model.save()`.
This file format is considered legacy. We recommend using instead the native
Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

In [ ]:

```
from tensorflow.keras.models import load_model
def RunTestHarness():
  trainX, trainY, testX, testY = LoadDataset()
  trainX, testX = prep_pixels(trainX, testX)
  model = load_model('/content/drive/MyDrive/CPE 019 (Retake)/Assignment
9.1/CNNmodel.h5')
  _, acc = model.evaluate(testX, testY, verbose=0)
  print('> %.3f' % (acc * 100.0))

RunTestHarness()
```

```
> 67.550
```

In [ ]:

```
from numpy import argmax
from tensorflow.keras.utils import load_img
from tensorflow.keras.utils import img_to_array
from keras.models import load_model

def load_image(filename):
  img = load_img(filename, grayscale=True, target_size=(32, 32))
  img = img_to_array(img)
  img = img.reshape(1, 32, 32, 3)
  img = img.astype('float32')
  img = img / 255.0
  return img

def run_example():
  (trainX, trainY), (testX, testY) = cifar10.load_data()
  trainX = trainX.astype('float32') / 255.0
  sample_image = trainX[0]
  sample_image = sample_image.reshape(1, 32, 32, 3)
  model = load_model('/content/drive/MyDrive/CPE 019 (Retake)/Assignment
9.1/CNNmodel.h5')
  predict_value = model.predict(sample_image)
  digit = argmax(predict_value)
  print(digit)

# entry point, run the example
run_example()
```

```
1/1 [==============================] - 0s 224ms/step
6
```

In [ ]:

```
from PIL import Image

def load_image(index):
    (X_train, _), (_, _) = cifar10.load_data()
    image = X_train[index]
    plt.imshow(image)
    plt.title(f"CIFAR-10 Image - Class: {index}")
    plt.axis('off')
    plt.show()

image_index = 0
load_image(image_index)
```

Observation: As seen in the result, The prediction is correct but the accuracy is low, with the percentage of 67%. There is a chance that it still might go wrong.

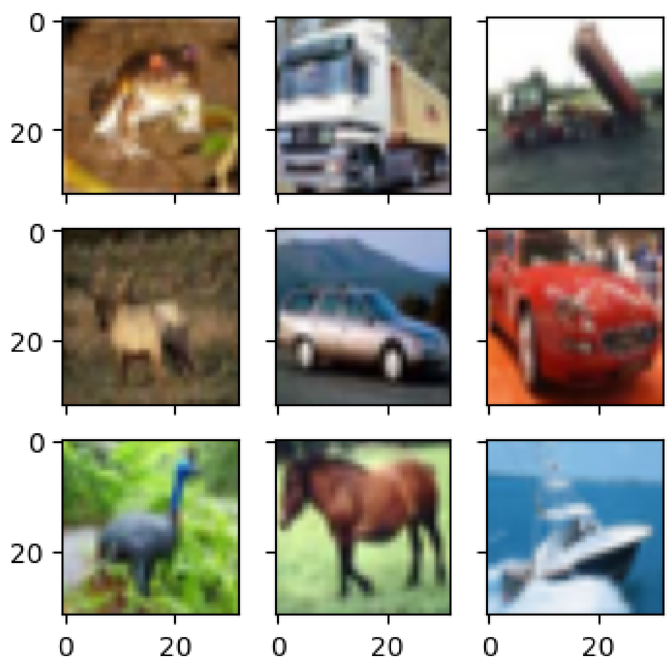
Perform image augmentation¶

In [ ]:

```
from tensorflow.keras.datasets import cifar10
import matplotlib.pyplot as plt
```

In [ ]:

```
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
fig, ax = plt.subplots(3, 3, sharex = True, sharey = True, figsize = (4, 4))
for i in range(3):
  for j in range(3):
    ax[i][j].imshow(X_train[i * 3 + j], cmap = plt.get_cmap('gray'))
plt.show()
```



Perform feature standardization¶

In [ ]:

```
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
```

In [ ]:

```
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
X_train = X_train.reshape(X_train.shape[0], 32, 32, 3)
X_test = X_test.reshape(X_test.shape[0], 32, 32, 3)

X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
```

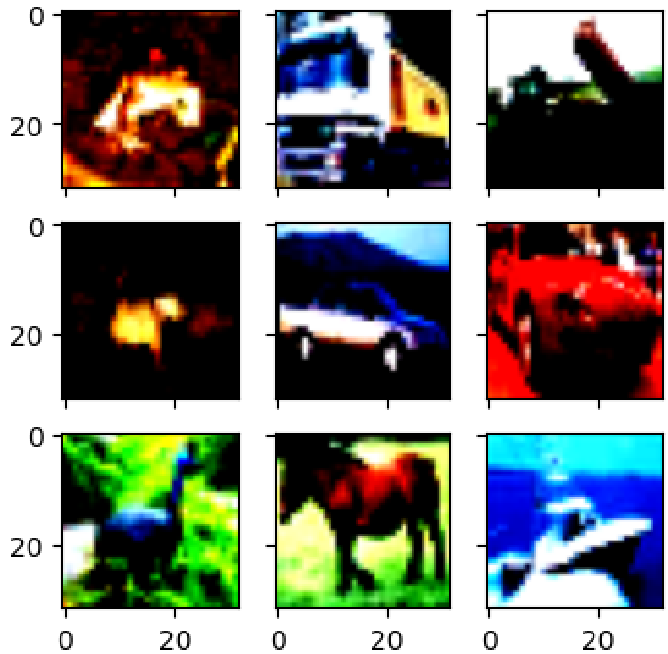
```
DataGen = ImageDataGenerator(featurewise_center = True,
featurewise_std_normalization = True)
```

In [ ]:

```
DataGen.fit(X_train)
for X_batch, y_batch in DataGen.flow(X_train, y_train, batch_size = 9,
shuffle = False):
  print(X_batch.min(), X_batch.mean(), X_batch.max())
  fig, ax = plt.subplots(3, 3, sharex = True, sharey = True, figsize = (4,
4))
  for i in range(3):
    for j in range(3):
      ax[i][j].imshow(X_batch[i * 3 + j], cmap = plt.get_cmap('gray'))
  plt.show()
  break
```

```
-1.9892113 -0.08899898 2.1267967
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow
with RGB data ([0..1] for floats or [0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow
with RGB data ([0..1] for floats or [0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow
with RGB data ([0..1] for floats or [0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow
with RGB data ([0..1] for floats or [0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow
with RGB data ([0..1] for floats or [0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow
with RGB data ([0..1] for floats or [0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow
with RGB data ([0..1] for floats or [0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow
with RGB data ([0..1] for floats or [0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow
with RGB data ([0..1] for floats or [0..255] for integers).
```



Observation: As seen in the result, the Feature Standardation has applied, the images is sharpen although the image is blurry.

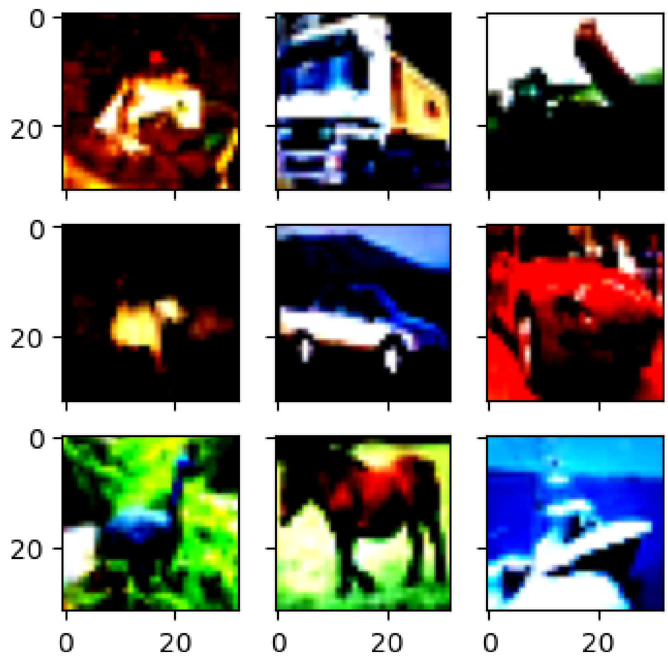
In [ ]:

```
DataGen.mean = X_train.mean(axis=0)
DataGen.std = X_train.std(axis=0)
for X_batch, y_batch in DataGen.flow(X_train, y_train, batch_size = 9,
shuffle = False):
  print(X_batch.min(), X_batch.mean(), X_batch.max())
  fig, ax = plt.subplots(3, 3, sharex = True, sharey = True, figsize = (4,
4))
  for i in range(3):
    for j in range(3):
      ax[i][j].imshow(X_batch[i * 3 + j], cmap = plt.get_cmap('gray'))
  plt.show()
  break
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow
with RGB data ([0..1] for floats or [0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow
with RGB data ([0..1] for floats or [0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow
with RGB data ([0..1] for floats or [0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow
with RGB data ([0..1] for floats or [0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow
with RGB data ([0..1] for floats or [0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow
```

```
with RGB data ([0..1] for floats or [0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow
with RGB data ([0..1] for floats or [0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow
with RGB data ([0..1] for floats or [0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow
with RGB data ([0..1] for floats or [0..255] for integers).
```

```
-2.002114 -0.09202043 2.5096273
```



Perform ZCA whitening of your images¶

In [ ]:

```
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
import numpy as np
```

In [ ]:

```
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
X_train = X_train.reshape(X_train.shape[0], 32, 32, 3)
X_test = X_test.reshape(X_test.shape[0], 32, 32, 3)
X_train = X_train.astype('float32') /255.0
X_test = X_test.astype('float32') / 255.0
DataGen = ImageDataGenerator(zca_whitening = True, featurewise_center = True,
featurewise_std_normalization = True)
```

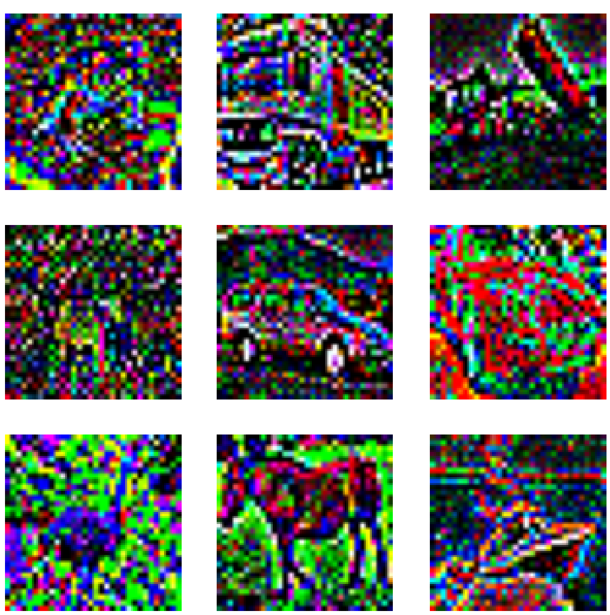
```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 4s 0us/step
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/preprocessing/image.py:1451
: UserWarning: This ImageDataGenerator specifies `zca_whitening` which
overrides setting of`featurewise_std_normalization`.
  warnings.warn(
```

In [ ]:

```
X_mean = X_train.mean(axis = 0)
DataGen.fit(X_train - X_mean)
print("check/n")
for X_batch, y_batch in DataGen.flow(X_train - X_mean, y_train, batch_size =
9, shuffle = False):
  print(X_batch.min(), X_batch.mean(), X_batch.max())
  fig, ax = plt.subplots(3, 3, sharex = True, sharey = True, figsize = (4,
4))
  for i in range(3):
    for j in range(3):
      img = np.clip(X_batch[i * 3 + j], 0, 1)
      ax[i][j].imshow(img)
      ax[i][j].axis('off')
  plt.show()
  break
```

```
check/n
-8.875162 -0.0023781403 8.505833
```



Observation: In the Result for this code, ZCA whitening is applied to the images, but for some reason, for i think it is not applied due to either distorted or it create color noise. I should make a manual or to see if it is still the same

In [1]:

```
from tensorflow.keras.datasets import cifar10
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:

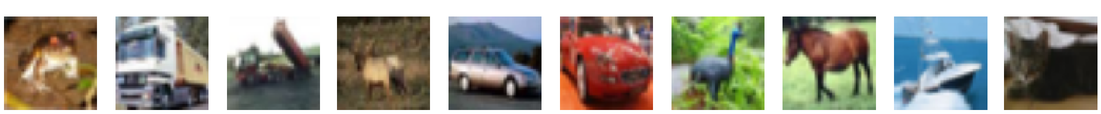
```
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0

flat_train_images = X_train.reshape(X_train.shape[0], -1)
sigma = np.dot(flat_train_images.T, flat_train_images) /
flat_train_images.shape[0]
U, S, V = np.linalg.svd(sigma)

epsilon = 1e-5
zca_matrix = np.dot(U, np.dot(np.diag(1.0 / np.sqrt(S + epsilon)), U.T))
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 4s 0us/step
```

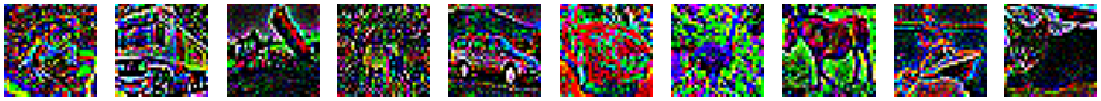
In [3]:

```
whitened_train_images = np.dot(flat_train_images, zca_matrix)
whitened_train_images = whitened_train_images.reshape(X_train.shape)

def plot_images(images, title):
    plt.figure(figsize=(10, 2))
    for i in range(10):
        plt.subplot(1, 10, i + 1)
        plt.imshow(np.clip(images[i], 0, 1))  # Clip the values to the range
[0, 1]
        plt.axis('off')
    plt.suptitle(title)
    plt.show()

plot_images(X_train, 'Original Images')
plot_images(whitened_train_images, 'ZCA Whitened Images')
```

Original Images



ZCA Whitened Images



Observation: This code is from ChatGPT for manualling whitening or manually zca whitening the images from cifar10, and it seems the zca whitening is applied in the previous code.

Augment data with random rotations, shifts, and flips¶

Random Rotation¶

In [ ]:

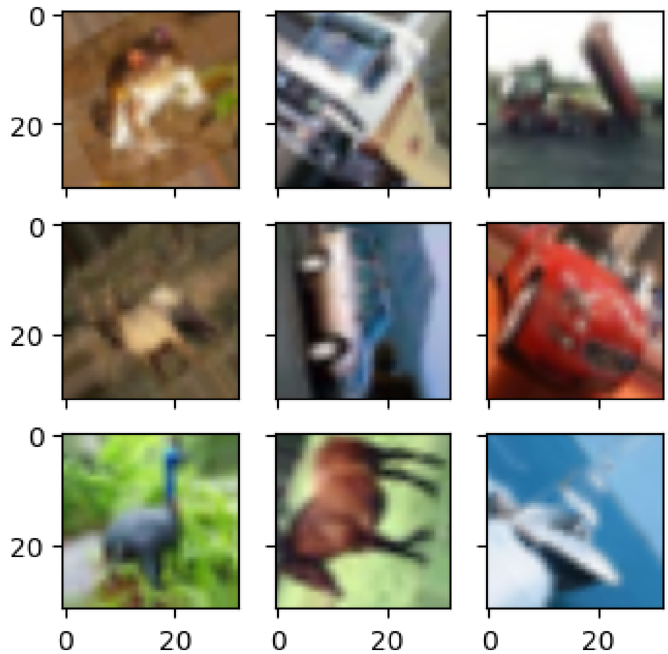
```
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
```

In [ ]:

```
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
X_train = X_train.reshape(X_train.shape[0], 32, 32, 3)
X_test = X_test.reshape(X_test.shape[0], 32, 32, 3)
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0
DataGen = ImageDataGenerator(rotation_range = 90)
```

In [ ]:

```
for X_batch, y_batch in DataGen.flow(X_train, y_train, batch_size = 9,
shuffle = False):
  fig, ax = plt.subplots(3, 3, sharex = True, sharey = True, figsize = (4,
4))
  for i in range(3):
    for j in range(3):
      ax[i][j].imshow(X_batch[i * 3 + j].reshape(32, 32, 3), cmap =
plt.get_cmap('gray'))
  plt.show()
  break
```



Observation: As seen in the result above, the code are succesfully implemented but for some reason the some images are the only ones that rotated 90 degrees while the other is random

Random Shifts¶

In [ ]:

```
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
```

In [ ]:

```
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
X_train = X_train.reshape(X_train.shape[0], 32, 32, 3)
X_test = X_test.reshape(X_test.shape[0], 32, 32, 3)
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0
shift = 0.2
DataGen = ImageDataGenerator(width_shift_range = shift, height_shift_range =
shift)
```

In [ ]:

```
for X_batch, y_batch in DataGen.flow(X_train, y_train, batch_size = 9,
shuffle = False):
  fig, ax = plt.subplots(3, 3, sharex = True, sharey = True, figsize = (4,
4))
  for i in range(3):
```

```
    for j in range(3):
      ax[i][j].imshow(X_batch[i * 3 + j].reshape(32, 32, 3), cmap = 
plt.get_cmap('gray'))
  plt.show()
  break
```

Observation: As seen in the result, all images is either slight elevated or shift but most of them is shifted to the random direction.

Random Flip¶

In [ ]:

```
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
```

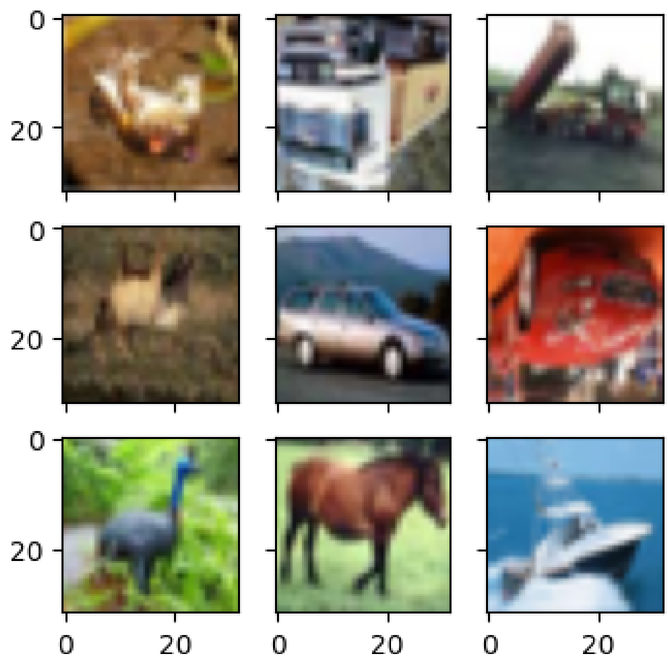
In [ ]:

```
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
X_train = X_train.reshape(X_train.shape[0], 32, 32, 3)
X_test = X_test.reshape(X_test.shape[0], 32, 32, 3)
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0
DataGen = ImageDataGenerator(horizontal_flip = True, vertical_flip = True)
```

In [ ]:

```
for X_batch, y_batch in DataGen.flow(X_train, y_train, batch_size = 9, 
shuffle = False):
```

```
  fig, ax = plt.subplots(3, 3, sharex = True, sharey = True, figsize = (4, 4))
  for i in range(3):
    for j in range(3):
      ax[i][j].imshow(X_batch[i * 3 + j].reshape(32, 32, 3), cmap = plt.get_cmap('gray'))
  plt.show()
  break
```



Observation: In this result, all images is flip randomly, either horizontal or vertical.

Save augmented image data to disk¶

In [ ]:

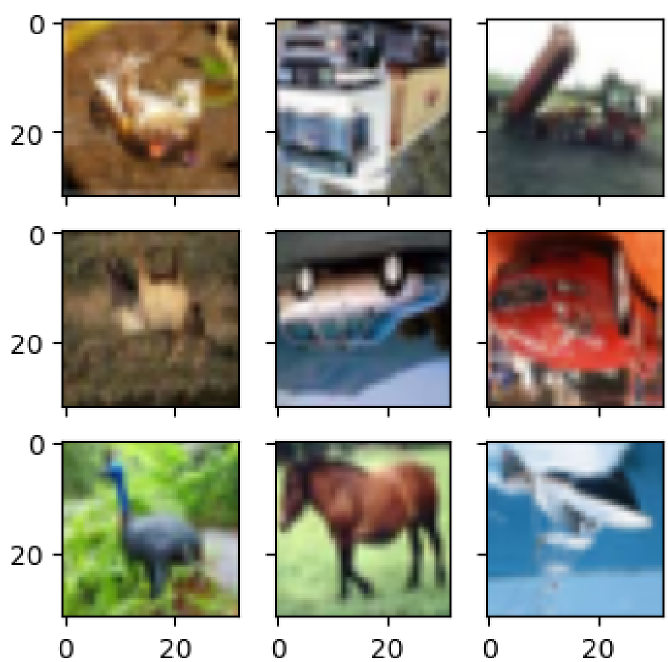
```
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
```

In [ ]:

```
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
X_train = X_train.reshape(X_train.shape[0], 32, 32, 3)
X_test = X_test.reshape(X_test.shape[0], 32, 32, 3)
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0
DataGen = ImageDataGenerator(horizontal_flip = True, vertical_flip = True)
```

In [ ]:

```
for X_batch, y_batch in DataGen.flow(X_train, y_train, batch_size = 9, 
shuffle = False, save_to_dir = '/content/drive/MyDrive/CPE 019 
(Retake)/Assignment 9.1',
                                     save_prefix = 'aug', save_format = 
'png'):
  fig, ax = plt.subplots(3, 3, sharex = True, sharey = True, figsize = (4, 
4))
  for i in range(3):
    for j in range(3):
      ax[i][j].imshow(X_batch[i * 3 + j].reshape(32, 32, 3), cmap = 
plt.get_cmap('gray'))
  plt.show()
  break
```



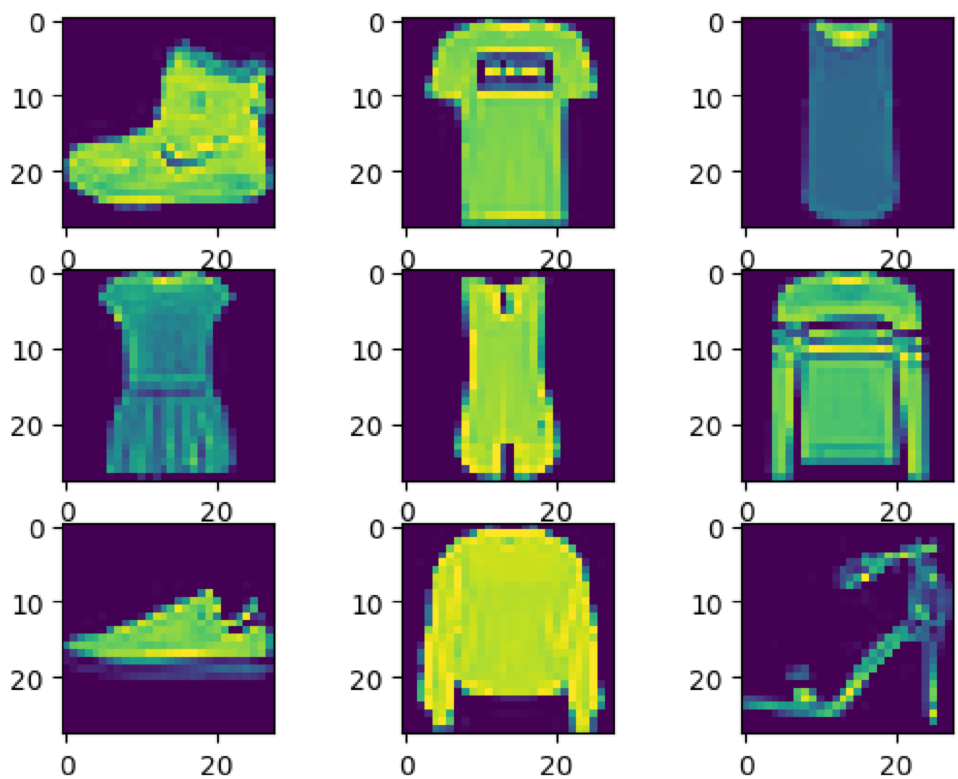
Develop a test harness to develop a robust evaluation of a model and establish a baseline of performance for a classification task¶

In [ ]:

```
from matplotlib import pyplot
from tensorflow.keras.datasets import fashion_mnist

(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()

print('Train: X=%s, y=%s' % (X_train.shape, y_train.shape))
print('Test: X=%s, y=%s' % (X_test.shape, y_test.shape))

for i in range(9):
```

```
  pyplot.subplot(330 + 1 + i)
  pyplot.imshow(X_train[i])

pyplot.show()
```

```
Downloading data from
https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1
-ubyte.gz
29515/29515 [==============================] - 0s 0us/step
Downloading data from
https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3
-ubyte.gz
26421880/26421880 [==============================] - 0s 0us/step
Downloading data from
https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-
ubyte.gz
5148/5148 [==============================] - 0s 0us/step
Downloading data from
https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-
ubyte.gz
4422102/4422102 [==============================] - 0s 0us/step
Train: X=(60000, 28, 28), y=(60000,)
Test: X=(10000, 28, 28), y=(10000,)
```



In [ ]:

```
import sys
from matplotlib import pyplot
from tensorflow.keras.datasets import fashion_mnist
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.optimizers import SGD
```

In [ ]:

```
def LoadDataset():
  (trainX, trainY), (testX, testY) = fashion_mnist.load_data()
  trainY= to_categorical(trainY)
  testY = to_categorical(testY)
  trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))
  testX = testX.reshape((testX.shape[0], 28, 28, 1))
  return trainX, trainY, testX, testY

def PrepPixels(Train, Test):
  train_norm = Train.astype('float32')
  test_norm = Test.astype('float32')
  train_norm = train_norm / 255.0
  test_norm = test_norm / 255.0
  return train_norm, test_norm
```

In [ ]:

```
def CNNModel():
  model = Sequential()
  model.add(Conv2D(32, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same', input_shape=(28, 28, 1)))
  model.add(MaxPooling2D((2, 2)))
  model.add(Flatten())
  model.add(Dense(20, activation='relu', kernel_initializer='he_uniform'))
  model.add(Dense(10, activation='softmax'))
  opt = SGD(lr=0.01, momentum=0.9)
  model.compile(optimizer=opt, loss='categorical_crossentropy',
metrics=['accuracy'])
  return model

def SummarizeDiagnostic(history):
  pyplot.subplot(211)
  pyplot.title('Cross Entropy Loss')
  pyplot.plot(history.history['loss'], color='blue', label='train')
  pyplot.plot(history.history['val_loss'], color='orange', label='test')

  pyplot.subplot(212)
```

```
  pyplot.title('Classification Accuracy')
  pyplot.plot(history.history['accuracy'], color='blue', label='train')
  pyplot.plot(history.history['val_accuracy'], color='orange', label='test')

  filename = sys.argv[0].split('/')[-1]
  pyplot.savefig(filename + '_plot.png')
  pyplot.close()
```

In [ ]:

```
def SummarizeHistory(histories):
  for i in range(len(histories)):
    plt.subplot(2, 1, 1)
    plt.title('Cross Entropy Loss')
    plt.plot(histories[i].history['loss'], color='blue', label='train')
    plt.plot(histories[i].history['val_loss'], color='orange', label='test')
    plt.subplot(2, 1, 2)
    plt.title('Classification Accuracy')
    plt.plot(histories[i].history['accuracy'], color='blue', label='train')
    plt.plot(histories[i].history['val_accuracy'], color='orange',
label='test')
  plt.show()
```

In [ ]:

```
def RunTestHarness2():
  (trainX, trainY, testX, testY) = LoadDataset()
  trainX, testX = PrepPixels(trainX, testX)
  model = CNNModel()
  history = model.fit(trainX, trainY, epochs=50, batch_size=500,
validation_data=(testX, testY), verbose=1)
  _, acc = model.evaluate(testX, testY, verbose=0)
  print('> %.3f' % (acc * 100.0))
  SummarizeDiagnostic(history)
```

In [ ]:

```
RunTestHarness2()
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use
`learning_rate` or use the legacy optimizer,
e.g.,tf.keras.optimizers.legacy.SGD.

Epoch 1/50
120/120 [==============================] - 37s 307ms/step - loss: 0.7164 -
accuracy: 0.7418 - val_loss: 0.4870 - val_accuracy: 0.8232
Epoch 2/50
120/120 [==============================] - 32s 266ms/step - loss: 0.4342 -
accuracy: 0.8487 - val_loss: 0.4242 - val_accuracy: 0.8469
Epoch 3/50
120/120 [==============================] - 31s 261ms/step - loss: 0.3907 -
```

```
accuracy: 0.8629 - val_loss: 0.4149 - val_accuracy: 0.8529
Epoch 4/50
120/120 [==============================] - 31s 261ms/step - loss: 0.3643 -
accuracy: 0.8731 - val_loss: 0.4017 - val_accuracy: 0.8555
Epoch 5/50
120/120 [==============================] - 32s 263ms/step - loss: 0.3557 -
accuracy: 0.8759 - val_loss: 0.3932 - val_accuracy: 0.8597
Epoch 6/50
120/120 [==============================] - 32s 264ms/step - loss: 0.3386 -
accuracy: 0.8816 - val_loss: 0.3625 - val_accuracy: 0.8740
Epoch 7/50
120/120 [==============================] - 31s 256ms/step - loss: 0.3191 -
accuracy: 0.8883 - val_loss: 0.3535 - val_accuracy: 0.8730
Epoch 8/50
120/120 [==============================] - 32s 267ms/step - loss: 0.3064 -
accuracy: 0.8928 - val_loss: 0.3277 - val_accuracy: 0.8849
Epoch 9/50
120/120 [==============================] - 32s 267ms/step - loss: 0.2990 -
accuracy: 0.8958 - val_loss: 0.3500 - val_accuracy: 0.8775
Epoch 10/50
120/120 [==============================] - 31s 259ms/step - loss: 0.2900 -
accuracy: 0.8990 - val_loss: 0.3218 - val_accuracy: 0.8854
Epoch 11/50
120/120 [==============================] - 31s 257ms/step - loss: 0.2853 -
accuracy: 0.8989 - val_loss: 0.3179 - val_accuracy: 0.8882
Epoch 12/50
120/120 [==============================] - 32s 269ms/step - loss: 0.2772 -
accuracy: 0.9020 - val_loss: 0.3089 - val_accuracy: 0.8904
Epoch 13/50
120/120 [==============================] - 32s 271ms/step - loss: 0.2694 -
accuracy: 0.9045 - val_loss: 0.3395 - val_accuracy: 0.8792
Epoch 14/50
120/120 [==============================] - 32s 263ms/step - loss: 0.2603 -
accuracy: 0.9075 - val_loss: 0.3144 - val_accuracy: 0.8833
Epoch 15/50
120/120 [==============================] - 32s 265ms/step - loss: 0.2565 -
accuracy: 0.9096 - val_loss: 0.3060 - val_accuracy: 0.8908
Epoch 16/50
120/120 [==============================] - 32s 267ms/step - loss: 0.2501 -
accuracy: 0.9118 - val_loss: 0.2941 - val_accuracy: 0.8931
Epoch 17/50
120/120 [==============================] - 32s 268ms/step - loss: 0.2433 -
accuracy: 0.9140 - val_loss: 0.2935 - val_accuracy: 0.8955
Epoch 18/50
120/120 [==============================] - 32s 269ms/step - loss: 0.2378 -
accuracy: 0.9164 - val_loss: 0.3147 - val_accuracy: 0.8891
Epoch 19/50
120/120 [==============================] - 31s 262ms/step - loss: 0.2373 -
accuracy: 0.9148 - val_loss: 0.2902 - val_accuracy: 0.8961
Epoch 20/50
```

```
120/120 [==============================] - 32s 270ms/step - loss: 0.2324 -
accuracy: 0.9173 - val_loss: 0.2843 - val_accuracy: 0.8981
Epoch 21/50
120/120 [==============================] - 31s 262ms/step - loss: 0.2284 -
accuracy: 0.9190 - val_loss: 0.2852 - val_accuracy: 0.8970
Epoch 22/50
120/120 [==============================] - 32s 270ms/step - loss: 0.2244 -
accuracy: 0.9207 - val_loss: 0.2831 - val_accuracy: 0.9000
Epoch 23/50
120/120 [==============================] - 32s 269ms/step - loss: 0.2209 -
accuracy: 0.9220 - val_loss: 0.2851 - val_accuracy: 0.8962
Epoch 24/50
120/120 [==============================] - 32s 268ms/step - loss: 0.2165 -
accuracy: 0.9239 - val_loss: 0.2781 - val_accuracy: 0.9008
Epoch 25/50
120/120 [==============================] - 32s 269ms/step - loss: 0.2112 -
accuracy: 0.9258 - val_loss: 0.2766 - val_accuracy: 0.8996
Epoch 26/50
120/120 [==============================] - 31s 262ms/step - loss: 0.2107 -
accuracy: 0.9257 - val_loss: 0.2870 - val_accuracy: 0.8933
Epoch 27/50
120/120 [==============================] - 31s 259ms/step - loss: 0.2072 -
accuracy: 0.9266 - val_loss: 0.2760 - val_accuracy: 0.9003
Epoch 28/50
120/120 [==============================] - 31s 263ms/step - loss: 0.2072 -
accuracy: 0.9271 - val_loss: 0.2914 - val_accuracy: 0.8935
Epoch 29/50
120/120 [==============================] - 33s 273ms/step - loss: 0.2010 -
accuracy: 0.9288 - val_loss: 0.2851 - val_accuracy: 0.8965
Epoch 30/50
120/120 [==============================] - 32s 264ms/step - loss: 0.1985 -
accuracy: 0.9308 - val_loss: 0.2680 - val_accuracy: 0.9058
Epoch 31/50
120/120 [==============================] - 32s 270ms/step - loss: 0.1971 -
accuracy: 0.9297 - val_loss: 0.2725 - val_accuracy: 0.9010
Epoch 32/50
120/120 [==============================] - 35s 287ms/step - loss: 0.1923 -
accuracy: 0.9326 - val_loss: 0.2708 - val_accuracy: 0.9028
Epoch 33/50
120/120 [==============================] - 32s 269ms/step - loss: 0.1882 -
accuracy: 0.9341 - val_loss: 0.2737 - val_accuracy: 0.9025
Epoch 34/50
120/120 [==============================] - 31s 260ms/step - loss: 0.1873 -
accuracy: 0.9347 - val_loss: 0.2695 - val_accuracy: 0.9030
Epoch 35/50
120/120 [==============================] - 31s 262ms/step - loss: 0.1837 -
accuracy: 0.9349 - val_loss: 0.2660 - val_accuracy: 0.9030
Epoch 36/50
120/120 [==============================] - 32s 269ms/step - loss: 0.1814 -
accuracy: 0.9359 - val_loss: 0.2689 - val_accuracy: 0.9055
```

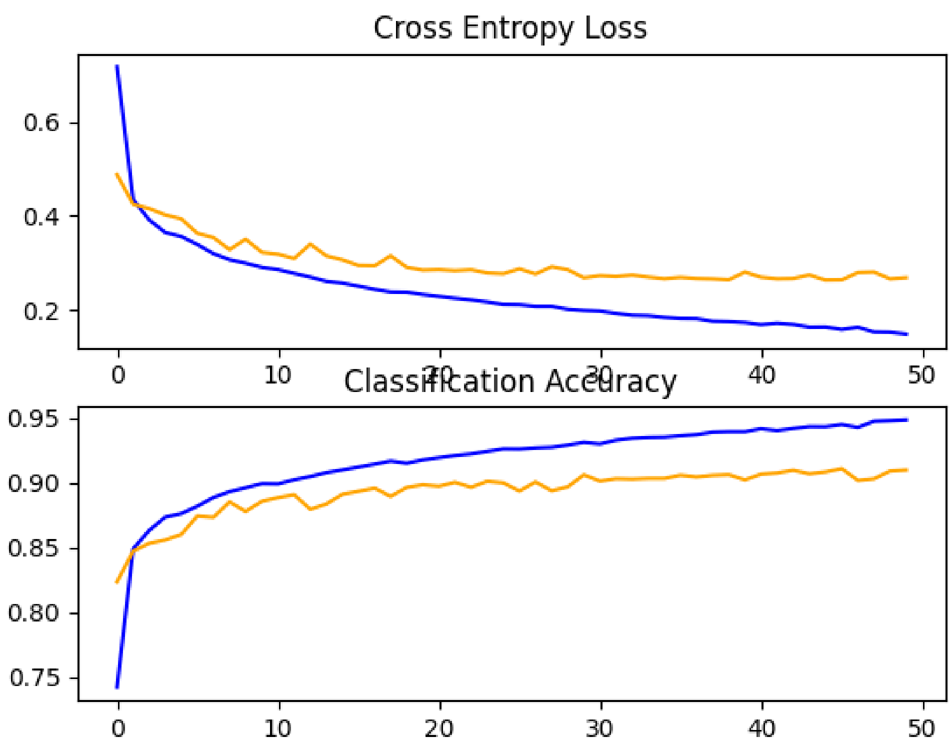
```
Epoch 37/50
120/120 [==============================] - 33s 273ms/step - loss: 0.1810 -
accuracy: 0.9369 - val_loss: 0.2664 - val_accuracy: 0.9042
Epoch 38/50
120/120 [==============================] - 35s 293ms/step - loss: 0.1755 -
accuracy: 0.9388 - val_loss: 0.2655 - val_accuracy: 0.9054
Epoch 39/50
120/120 [==============================] - 32s 270ms/step - loss: 0.1746 -
accuracy: 0.9391 - val_loss: 0.2636 - val_accuracy: 0.9059
Epoch 40/50
120/120 [==============================] - 32s 268ms/step - loss: 0.1728 -
accuracy: 0.9391 - val_loss: 0.2800 - val_accuracy: 0.9019
Epoch 41/50
120/120 [==============================] - 32s 270ms/step - loss: 0.1684 -
accuracy: 0.9414 - val_loss: 0.2692 - val_accuracy: 0.9063
Epoch 42/50
120/120 [==============================] - 31s 262ms/step - loss: 0.1713 -
accuracy: 0.9400 - val_loss: 0.2658 - val_accuracy: 0.9072
Epoch 43/50
120/120 [==============================] - 32s 270ms/step - loss: 0.1687 -
accuracy: 0.9416 - val_loss: 0.2665 - val_accuracy: 0.9093
Epoch 44/50
120/120 [==============================] - 35s 288ms/step - loss: 0.1631 -
accuracy: 0.9431 - val_loss: 0.2739 - val_accuracy: 0.9066
Epoch 45/50
120/120 [==============================] - 31s 262ms/step - loss: 0.1634 -
accuracy: 0.9430 - val_loss: 0.2630 - val_accuracy: 0.9080
Epoch 46/50
120/120 [==============================] - 32s 270ms/step - loss: 0.1581 -
accuracy: 0.9446 - val_loss: 0.2634 - val_accuracy: 0.9106
Epoch 47/50
120/120 [==============================] - 31s 259ms/step - loss: 0.1628 -
accuracy: 0.9424 - val_loss: 0.2789 - val_accuracy: 0.9015
Epoch 48/50
120/120 [==============================] - 32s 267ms/step - loss: 0.1528 -
accuracy: 0.9472 - val_loss: 0.2800 - val_accuracy: 0.9027
Epoch 49/50
120/120 [==============================] - 31s 261ms/step - loss: 0.1523 -
accuracy: 0.9477 - val_loss: 0.2657 - val_accuracy: 0.9087
Epoch 50/50
120/120 [==============================] - 32s 270ms/step - loss: 0.1483 -
accuracy: 0.9483 - val_loss: 0.2682 - val_accuracy: 0.9095
> 90.950
```

In [ ]:

```
from PIL import Image

def load_image(file_path):
    """
```

```
    Load an image file and return a PIL Image object.

    Args:
        file_path (str): The file path of the image file.

    Returns:
        Image: A PIL Image object.
    """
    try:
        image = Image.open(file_path)
        return image
    except IOError:
        print("Unable to load image")
        return None
load_image('/content/colab_kernel_launcher.py_plot.png')
```

Out[ ]:



Observation: In this part of code, it execute too long, it takes several minutes to finish the execution, the accuracy is 90% percent and it is great but it can improve more.

Explore extensions to a baseline model to improve learning and model capacity.¶

In [ ]:

```
import time
start_time = time.time()
import sys
from matplotlib import pyplot
from tensorflow.keras.datasets import fashion_mnist
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.optimizers import SGD
from keras.layers import Dropout
```

In [ ]:

```
def LoadDataset():
  (trainX, trainY), (testX, testY) = fashion_mnist.load_data()
  trainY= to_categorical(trainY)
  testY = to_categorical(testY)
  trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))
  testX = testX.reshape((testX.shape[0], 28, 28, 1))
  return trainX, trainY, testX, testY

def PrepPixels(Train, Test):
  train_norm = Train.astype('float32')
  test_norm = Test.astype('float32')
  train_norm = train_norm / 255.0
  test_norm = test_norm / 255.0
  return train_norm, test_norm
```

In [ ]:

```
def CNNModel():
  model = Sequential()
  model.add(Conv2D(32, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same', input_shape=(28, 28, 1)))
  model.add(Conv2D(32, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same'))
  model.add(MaxPooling2D((2, 2)))
  model.add(Flatten())
  model.add(Dense(40, activation='relu', kernel_initializer='he_uniform'))
  model.add(Dropout(0.2))
  model.add(Dense(30, activation='relu', kernel_initializer='he_uniform'))
  model.add(Dropout(0.2))
```

```
  model.add(Dense(10, activation='softmax'))
  opt = SGD(lr=0.01, momentum=0.9)
  model.compile(optimizer=opt, loss='categorical_crossentropy',
metrics=['accuracy'])
  return model

def SummarizeDiagnostic(history):
  pyplot.subplot(211)
  pyplot.title('Cross Entropy Loss')
  pyplot.plot(history.history['loss'], color='blue', label='train')
  pyplot.plot(history.history['val_loss'], color='orange', label='test')

  pyplot.subplot(212)
  pyplot.title('Classification Accuracy')
  pyplot.plot(history.history['accuracy'], color='blue', label='train')
  pyplot.plot(history.history['val_accuracy'], color='orange', label='test')

  filename = sys.argv[0].split('/')[-1]
  pyplot.savefig(filename + '_plot.png')
  pyplot.close()
```

In [ ]:

```
def RunTestHarness2():
  (trainX, trainY, testX, testY) = LoadDataset()
  trainX, testX = PrepPixels(trainX, testX)
  model = CNNModel()
  history = model.fit(trainX, trainY, epochs=50, batch_size=328,
validation_data=(testX, testY), verbose=1)
  _, acc = model.evaluate(testX, testY, verbose=0)
  print('> %.3f' % (acc * 100.0))
  SummarizeDiagnostic(history)
```

In [ ]:

```
RunTestHarness2()
end_time = time.time()

total_time = end_time - start_time
print("Total time taken:", total_time/60, "minutes")
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use
`learning_rate` or use the legacy optimizer,
e.g.,tf.keras.optimizers.legacy.SGD.
```

```
Epoch 1/50
183/183 [==============================] - 121s 658ms/step - loss: 0.9413 -
accuracy: 0.6577 - val_loss: 0.4855 - val_accuracy: 0.8307
Epoch 2/50
183/183 [==============================] - 122s 665ms/step - loss: 0.5681 -
accuracy: 0.7997 - val_loss: 0.4287 - val_accuracy: 0.8467
```

```
Epoch 3/50
183/183 [==============================] - 121s 659ms/step - loss: 0.4909 -
accuracy: 0.8284 - val_loss: 0.3819 - val_accuracy: 0.8640
Epoch 4/50
183/183 [==============================] - 122s 666ms/step - loss: 0.4427 -
accuracy: 0.8460 - val_loss: 0.3539 - val_accuracy: 0.8769
Epoch 5/50
183/183 [==============================] - 124s 680ms/step - loss: 0.4052 -
accuracy: 0.8598 - val_loss: 0.3328 - val_accuracy: 0.8823
Epoch 6/50
183/183 [==============================] - 121s 661ms/step - loss: 0.3762 -
accuracy: 0.8688 - val_loss: 0.3242 - val_accuracy: 0.8852
Epoch 7/50
183/183 [==============================] - 121s 658ms/step - loss: 0.3558 -
accuracy: 0.8762 - val_loss: 0.3149 - val_accuracy: 0.8901
Epoch 8/50
183/183 [==============================] - 127s 693ms/step - loss: 0.3413 -
accuracy: 0.8806 - val_loss: 0.3042 - val_accuracy: 0.8917
Epoch 9/50
183/183 [==============================] - 127s 694ms/step - loss: 0.3244 -
accuracy: 0.8861 - val_loss: 0.3083 - val_accuracy: 0.8901
Epoch 10/50
183/183 [==============================] - 124s 679ms/step - loss: 0.3115 -
accuracy: 0.8906 - val_loss: 0.2923 - val_accuracy: 0.8975
Epoch 11/50
183/183 [==============================] - 123s 675ms/step - loss: 0.2960 -
accuracy: 0.8960 - val_loss: 0.2838 - val_accuracy: 0.9003
Epoch 12/50
183/183 [==============================] - 128s 702ms/step - loss: 0.2865 -
accuracy: 0.8982 - val_loss: 0.2843 - val_accuracy: 0.9000
Epoch 13/50
183/183 [==============================] - 122s 667ms/step - loss: 0.2803 -
accuracy: 0.9019 - val_loss: 0.2793 - val_accuracy: 0.9021
Epoch 14/50
183/183 [==============================] - 124s 680ms/step - loss: 0.2690 -
accuracy: 0.9054 - val_loss: 0.2881 - val_accuracy: 0.8986
Epoch 15/50
183/183 [==============================] - 124s 676ms/step - loss: 0.2594 -
accuracy: 0.9088 - val_loss: 0.2755 - val_accuracy: 0.9050
Epoch 16/50
183/183 [==============================] - 121s 660ms/step - loss: 0.2543 -
accuracy: 0.9108 - val_loss: 0.2716 - val_accuracy: 0.9058
Epoch 17/50
183/183 [==============================] - 123s 675ms/step - loss: 0.2466 -
accuracy: 0.9120 - val_loss: 0.2681 - val_accuracy: 0.9090
Epoch 18/50
183/183 [==============================] - 122s 666ms/step - loss: 0.2389 -
accuracy: 0.9149 - val_loss: 0.2734 - val_accuracy: 0.9064
Epoch 19/50
183/183 [==============================] - 121s 662ms/step - loss: 0.2329 -
```

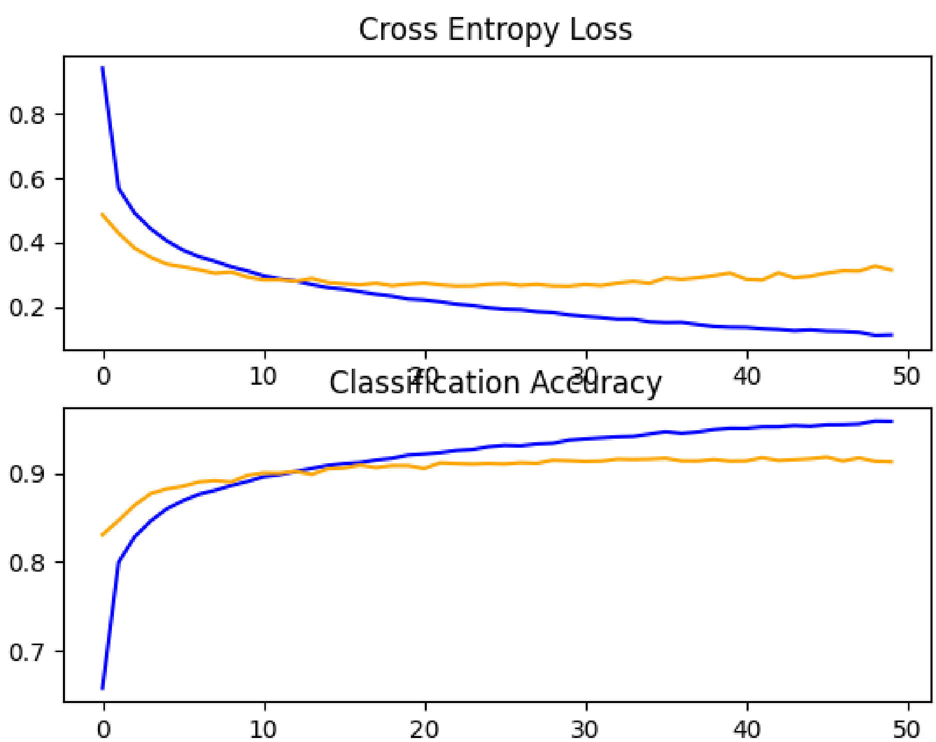
```
accuracy: 0.9168 - val_loss: 0.2655 - val_accuracy: 0.9085
Epoch 20/50
183/183 [==============================] - 124s 680ms/step - loss: 0.2240 -
accuracy: 0.9204 - val_loss: 0.2702 - val_accuracy: 0.9084
Epoch 21/50
183/183 [==============================] - 124s 679ms/step - loss: 0.2212 -
accuracy: 0.9216 - val_loss: 0.2737 - val_accuracy: 0.9053
Epoch 22/50
183/183 [==============================] - 124s 676ms/step - loss: 0.2152 -
accuracy: 0.9232 - val_loss: 0.2680 - val_accuracy: 0.9117
Epoch 23/50
183/183 [==============================] - 122s 668ms/step - loss: 0.2081 -
accuracy: 0.9256 - val_loss: 0.2645 - val_accuracy: 0.9109
Epoch 24/50
183/183 [==============================] - 125s 681ms/step - loss: 0.2035 -
accuracy: 0.9267 - val_loss: 0.2651 - val_accuracy: 0.9104
Epoch 25/50
183/183 [==============================] - 123s 671ms/step - loss: 0.1968 -
accuracy: 0.9298 - val_loss: 0.2699 - val_accuracy: 0.9110
Epoch 26/50
183/183 [==============================] - 118s 643ms/step - loss: 0.1921 -
accuracy: 0.9314 - val_loss: 0.2719 - val_accuracy: 0.9105
Epoch 27/50
183/183 [==============================] - 122s 666ms/step - loss: 0.1906 -
accuracy: 0.9309 - val_loss: 0.2664 - val_accuracy: 0.9118
Epoch 28/50
183/183 [==============================] - 123s 675ms/step - loss: 0.1847 -
accuracy: 0.9329 - val_loss: 0.2696 - val_accuracy: 0.9112
Epoch 29/50
183/183 [==============================] - 121s 663ms/step - loss: 0.1818 -
accuracy: 0.9337 - val_loss: 0.2648 - val_accuracy: 0.9146
Epoch 30/50
183/183 [==============================] - 127s 696ms/step - loss: 0.1748 -
accuracy: 0.9373 - val_loss: 0.2639 - val_accuracy: 0.9140
Epoch 31/50
183/183 [==============================] - 122s 668ms/step - loss: 0.1699 -
accuracy: 0.9387 - val_loss: 0.2692 - val_accuracy: 0.9132
Epoch 32/50
183/183 [==============================] - 124s 680ms/step - loss: 0.1664 -
accuracy: 0.9398 - val_loss: 0.2658 - val_accuracy: 0.9136
Epoch 33/50
183/183 [==============================] - 125s 681ms/step - loss: 0.1613 -
accuracy: 0.9409 - val_loss: 0.2736 - val_accuracy: 0.9159
Epoch 34/50
183/183 [==============================] - 127s 694ms/step - loss: 0.1613 -
accuracy: 0.9415 - val_loss: 0.2787 - val_accuracy: 0.9155
Epoch 35/50
183/183 [==============================] - 124s 678ms/step - loss: 0.1531 -
accuracy: 0.9441 - val_loss: 0.2732 - val_accuracy: 0.9159
Epoch 36/50
```

```
183/183 [==============================] - 126s 691ms/step - loss: 0.1508 -
accuracy: 0.9468 - val_loss: 0.2899 - val_accuracy: 0.9168
Epoch 37/50
183/183 [==============================] - 122s 666ms/step - loss: 0.1513 -
accuracy: 0.9449 - val_loss: 0.2853 - val_accuracy: 0.9138
Epoch 38/50
183/183 [==============================] - 119s 652ms/step - loss: 0.1447 -
accuracy: 0.9465 - val_loss: 0.2900 - val_accuracy: 0.9137
Epoch 39/50
183/183 [==============================] - 121s 661ms/step - loss: 0.1386 -
accuracy: 0.9492 - val_loss: 0.2967 - val_accuracy: 0.9155
Epoch 40/50
183/183 [==============================] - 119s 647ms/step - loss: 0.1364 -
accuracy: 0.9507 - val_loss: 0.3045 - val_accuracy: 0.9136
Epoch 41/50
183/183 [==============================] - 120s 657ms/step - loss: 0.1358 -
accuracy: 0.9506 - val_loss: 0.2858 - val_accuracy: 0.9138
Epoch 42/50
183/183 [==============================] - 121s 661ms/step - loss: 0.1319 -
accuracy: 0.9522 - val_loss: 0.2833 - val_accuracy: 0.9175
Epoch 43/50
183/183 [==============================] - 132s 718ms/step - loss: 0.1294 -
accuracy: 0.9523 - val_loss: 0.3050 - val_accuracy: 0.9144
Epoch 44/50
183/183 [==============================] - 148s 807ms/step - loss: 0.1255 -
accuracy: 0.9536 - val_loss: 0.2900 - val_accuracy: 0.9153
Epoch 45/50
183/183 [==============================] - 131s 714ms/step - loss: 0.1280 -
accuracy: 0.9529 - val_loss: 0.2949 - val_accuracy: 0.9163
Epoch 46/50
183/183 [==============================] - 123s 671ms/step - loss: 0.1245 -
accuracy: 0.9546 - val_loss: 0.3047 - val_accuracy: 0.9181
Epoch 47/50
183/183 [==============================] - 126s 687ms/step - loss: 0.1233 -
accuracy: 0.9548 - val_loss: 0.3116 - val_accuracy: 0.9138
Epoch 48/50
183/183 [==============================] - 127s 693ms/step - loss: 0.1206 -
accuracy: 0.9554 - val_loss: 0.3110 - val_accuracy: 0.9172
Epoch 49/50
183/183 [==============================] - 123s 673ms/step - loss: 0.1112 -
accuracy: 0.9588 - val_loss: 0.3263 - val_accuracy: 0.9134
Epoch 50/50
183/183 [==============================] - 126s 689ms/step - loss: 0.1126 -
accuracy: 0.9585 - val_loss: 0.3145 - val_accuracy: 0.9128
> 91.280
Total time taken: 103.45425353447597 minutes
```

In [ ]:

```
from PIL import Image

def load_image(file_path):
    """
    Load an image file and return a PIL Image object.

    Args:
        file_path (str): The file path of the image file.

    Returns:
        Image: A PIL Image object.
    """
    try:
        image = Image.open(file_path)
        return image
    except IOError:
        print("Unable to load image")
        return None
load_image('/content/colab_kernel_launcher.py_plot.png')
```

Out[ ]:



Observation: In this part of the code, the excution take 1 hour 43 minutes and 22 seconds, it takes too long to execute and I do not know the reason, the accuracy improve a little by 1.28%.

Develop a finalized model, evaluate the performance of the final model, and use it to make predictions on new images.¶

In [1]:

```
import time
start_time = time.time()
import sys
from matplotlib import pyplot
from tensorflow.keras.datasets import fashion_mnist
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.optimizers import SGD
from keras.layers import Dropout
```

In [2]:

```
def LoadDataset():
  (trainX, trainY), (testX, testY) = fashion_mnist.load_data()
  trainY= to_categorical(trainY)
  testY = to_categorical(testY)
  trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))
  testX = testX.reshape((testX.shape[0], 28, 28, 1))
  return trainX, trainY, testX, testY

def PrepPixels(Train, Test):
  train_norm = Train.astype('float32')
  test_norm = Test.astype('float32')
  train_norm = train_norm / 255.0
  test_norm = test_norm / 255.0
  return train_norm, test_norm
```

In [3]:

```
def CNNModel():
  model = Sequential()
  model.add(Conv2D(64, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same', input_shape=(28, 28, 1)))
  model.add(Conv2D(64, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same'))
  model.add(MaxPooling2D((2, 2)))
  model.add(Conv2D(32, (3, 3), activation='relu',
```

```
kernel_initializer='he_uniform', padding='same'))
  model.add(Conv2D(32, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same'))
  model.add(MaxPooling2D((2, 2)))
  model.add(Flatten())
  model.add(Dense(85, activation='relu', kernel_initializer='he_uniform'))
  model.add(Dropout(0.2))
  model.add(Dense(75, activation='relu', kernel_initializer='he_uniform'))
  model.add(Dropout(0.2))
  model.add(Dense(75, activation='relu', kernel_initializer='he_uniform'))
  model.add(Dropout(0.2))
  model.add(Dense(10, activation='softmax'))
  opt = SGD(lr=0.01, momentum=0.9)
  model.compile(optimizer=opt, loss='categorical_crossentropy',
metrics=['accuracy'])
  return model
```

In [4]:

```
def RunTestHarness2():
  (trainX, trainY, testX, testY) = LoadDataset()
  trainX, testX = PrepPixels(trainX, testX)
  model = CNNModel()
  history = model.fit(trainX, trainY, epochs=50, batch_size=500,
validation_data=(testX, testY), verbose=1)
  model.save('/content/drive/MyDrive/CPE 019 (Retake)/Assignment
9.1/final_model.h5')
```

In [5]:

```
RunTestHarness2()
end_time = time.time()

total_time = end_time - start_time
print("Total time taken:", total_time/60, "minutes")
```

```
Downloading data from
https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1
-ubyte.gz
29515/29515 [==============================] - 0s 0us/step
Downloading data from
https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3
-ubyte.gz
26421880/26421880 [==============================] - 1s 0us/step
Downloading data from
https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-
ubyte.gz
5148/5148 [==============================] - 0s 0us/step
Downloading data from
https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-
```

```
ubyte.gz
4422102/4422102 [==============================] - 1s 0us/step

WARNING:absl:`lr` is deprecated in Keras optimizer, please use
`learning_rate` or use the legacy optimizer,
e.g.,tf.keras.optimizers.legacy.SGD.

Epoch 1/50
120/120 [==============================] - 11s 38ms/step - loss: 1.1174 -
accuracy: 0.5829 - val_loss: 0.5350 - val_accuracy: 0.7998
Epoch 2/50
120/120 [==============================] - 4s 34ms/step - loss: 0.5971 -
accuracy: 0.7856 - val_loss: 0.4482 - val_accuracy: 0.8394
Epoch 3/50
120/120 [==============================] - 4s 36ms/step - loss: 0.5070 -
accuracy: 0.8229 - val_loss: 0.4039 - val_accuracy: 0.8517
Epoch 4/50
120/120 [==============================] - 4s 37ms/step - loss: 0.4546 -
accuracy: 0.8406 - val_loss: 0.3701 - val_accuracy: 0.8659
Epoch 5/50
120/120 [==============================] - 4s 36ms/step - loss: 0.4119 -
accuracy: 0.8552 - val_loss: 0.3594 - val_accuracy: 0.8683
Epoch 6/50
120/120 [==============================] - 4s 36ms/step - loss: 0.3920 -
accuracy: 0.8638 - val_loss: 0.3406 - val_accuracy: 0.8785
Epoch 7/50
120/120 [==============================] - 4s 36ms/step - loss: 0.3656 -
accuracy: 0.8724 - val_loss: 0.3252 - val_accuracy: 0.8851
Epoch 8/50
120/120 [==============================] - 4s 36ms/step - loss: 0.3488 -
accuracy: 0.8790 - val_loss: 0.3209 - val_accuracy: 0.8829
Epoch 9/50
120/120 [==============================] - 4s 37ms/step - loss: 0.3386 -
accuracy: 0.8818 - val_loss: 0.2987 - val_accuracy: 0.8929
Epoch 10/50
120/120 [==============================] - 5s 38ms/step - loss: 0.3190 -
accuracy: 0.8888 - val_loss: 0.2975 - val_accuracy: 0.8946
Epoch 11/50
120/120 [==============================] - 4s 36ms/step - loss: 0.3042 -
accuracy: 0.8924 - val_loss: 0.3015 - val_accuracy: 0.8924
Epoch 12/50
120/120 [==============================] - 4s 37ms/step - loss: 0.2959 -
accuracy: 0.8979 - val_loss: 0.2807 - val_accuracy: 0.9019
Epoch 13/50
120/120 [==============================] - 5s 38ms/step - loss: 0.2867 -
accuracy: 0.9004 - val_loss: 0.2808 - val_accuracy: 0.9010
Epoch 14/50
120/120 [==============================] - 5s 38ms/step - loss: 0.2808 -
accuracy: 0.9013 - val_loss: 0.2737 - val_accuracy: 0.9034
Epoch 15/50
```

```
120/120 [==============================] - 4s 37ms/step - loss: 0.2700 -
accuracy: 0.9061 - val_loss: 0.2641 - val_accuracy: 0.9074
Epoch 16/50
120/120 [==============================] - 5s 38ms/step - loss: 0.2623 -
accuracy: 0.9092 - val_loss: 0.2695 - val_accuracy: 0.9051
Epoch 17/50
120/120 [==============================] - 4s 37ms/step - loss: 0.2573 -
accuracy: 0.9098 - val_loss: 0.2637 - val_accuracy: 0.9090
Epoch 18/50
120/120 [==============================] - 4s 37ms/step - loss: 0.2521 -
accuracy: 0.9112 - val_loss: 0.2673 - val_accuracy: 0.9083
Epoch 19/50
120/120 [==============================] - 5s 43ms/step - loss: 0.2426 -
accuracy: 0.9160 - val_loss: 0.2644 - val_accuracy: 0.9067
Epoch 20/50
120/120 [==============================] - 4s 37ms/step - loss: 0.2417 -
accuracy: 0.9146 - val_loss: 0.2516 - val_accuracy: 0.9088
Epoch 21/50
120/120 [==============================] - 4s 36ms/step - loss: 0.2298 -
accuracy: 0.9188 - val_loss: 0.2566 - val_accuracy: 0.9114
Epoch 22/50
120/120 [==============================] - 4s 37ms/step - loss: 0.2253 -
accuracy: 0.9216 - val_loss: 0.2599 - val_accuracy: 0.9109
Epoch 23/50
120/120 [==============================] - 4s 37ms/step - loss: 0.2218 -
accuracy: 0.9217 - val_loss: 0.2510 - val_accuracy: 0.9131
Epoch 24/50
120/120 [==============================] - 4s 37ms/step - loss: 0.2161 -
accuracy: 0.9239 - val_loss: 0.2519 - val_accuracy: 0.9166
Epoch 25/50
120/120 [==============================] - 4s 37ms/step - loss: 0.2114 -
accuracy: 0.9248 - val_loss: 0.2505 - val_accuracy: 0.9140
Epoch 26/50
120/120 [==============================] - 4s 37ms/step - loss: 0.2084 -
accuracy: 0.9260 - val_loss: 0.2497 - val_accuracy: 0.9169
Epoch 27/50
120/120 [==============================] - 4s 37ms/step - loss: 0.2043 -
accuracy: 0.9279 - val_loss: 0.2463 - val_accuracy: 0.9183
Epoch 28/50
120/120 [==============================] - 5s 38ms/step - loss: 0.1988 -
accuracy: 0.9295 - val_loss: 0.2496 - val_accuracy: 0.9140
Epoch 29/50
120/120 [==============================] - 4s 37ms/step - loss: 0.1967 -
accuracy: 0.9305 - val_loss: 0.2547 - val_accuracy: 0.9108
Epoch 30/50
120/120 [==============================] - 4s 37ms/step - loss: 0.1935 -
accuracy: 0.9315 - val_loss: 0.2467 - val_accuracy: 0.9168
Epoch 31/50
120/120 [==============================] - 5s 38ms/step - loss: 0.1883 -
accuracy: 0.9327 - val_loss: 0.2571 - val_accuracy: 0.9152
```

```
Epoch 32/50
120/120 [==============================] - 4s 37ms/step - loss: 0.1832 -
accuracy: 0.9349 - val_loss: 0.2552 - val_accuracy: 0.9154
Epoch 33/50
120/120 [==============================] - 4s 37ms/step - loss: 0.1780 -
accuracy: 0.9368 - val_loss: 0.2402 - val_accuracy: 0.9187
Epoch 34/50
120/120 [==============================] - 4s 37ms/step - loss: 0.1780 -
accuracy: 0.9365 - val_loss: 0.2553 - val_accuracy: 0.9190
Epoch 35/50
120/120 [==============================] - 4s 36ms/step - loss: 0.1749 -
accuracy: 0.9374 - val_loss: 0.2442 - val_accuracy: 0.9205
Epoch 36/50
120/120 [==============================] - 4s 37ms/step - loss: 0.1686 -
accuracy: 0.9405 - val_loss: 0.2529 - val_accuracy: 0.9195
Epoch 37/50
120/120 [==============================] - 5s 38ms/step - loss: 0.1651 -
accuracy: 0.9407 - val_loss: 0.2578 - val_accuracy: 0.9165
Epoch 38/50
120/120 [==============================] - 4s 37ms/step - loss: 0.1664 -
accuracy: 0.9407 - val_loss: 0.2541 - val_accuracy: 0.9198
Epoch 39/50
120/120 [==============================] - 4s 37ms/step - loss: 0.1628 -
accuracy: 0.9411 - val_loss: 0.2487 - val_accuracy: 0.9182
Epoch 40/50
120/120 [==============================] - 5s 38ms/step - loss: 0.1577 -
accuracy: 0.9436 - val_loss: 0.2494 - val_accuracy: 0.9222
Epoch 41/50
120/120 [==============================] - 4s 36ms/step - loss: 0.1513 -
accuracy: 0.9456 - val_loss: 0.2453 - val_accuracy: 0.9212
Epoch 42/50
120/120 [==============================] - 4s 37ms/step - loss: 0.1531 -
accuracy: 0.9440 - val_loss: 0.2491 - val_accuracy: 0.9235
Epoch 43/50
120/120 [==============================] - 4s 37ms/step - loss: 0.1468 -
accuracy: 0.9478 - val_loss: 0.2601 - val_accuracy: 0.9208
Epoch 44/50
120/120 [==============================] - 4s 36ms/step - loss: 0.1470 -
accuracy: 0.9474 - val_loss: 0.2444 - val_accuracy: 0.9213
Epoch 45/50
120/120 [==============================] - 4s 37ms/step - loss: 0.1421 -
accuracy: 0.9495 - val_loss: 0.2472 - val_accuracy: 0.9239
Epoch 46/50
120/120 [==============================] - 5s 38ms/step - loss: 0.1401 -
accuracy: 0.9505 - val_loss: 0.2566 - val_accuracy: 0.9210
Epoch 47/50
120/120 [==============================] - 4s 36ms/step - loss: 0.1367 -
accuracy: 0.9511 - val_loss: 0.2552 - val_accuracy: 0.9207
Epoch 48/50
120/120 [==============================] - 4s 36ms/step - loss: 0.1340 -
```

```
accuracy: 0.9516 - val_loss: 0.2596 - val_accuracy: 0.9189
Epoch 49/50
120/120 [==============================] - 5s 38ms/step - loss: 0.1334 -
accuracy: 0.9529 - val_loss: 0.2498 - val_accuracy: 0.9224
Epoch 50/50
120/120 [==============================] - 4s 36ms/step - loss: 0.1309 -
accuracy: 0.9538 - val_loss: 0.2602 - val_accuracy: 0.9225
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103:
UserWarning: You are saving your model as an HDF5 file via `model.save()`.
This file format is considered legacy. We recommend using instead the native
Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

```
Total time taken: 4.197998615105947 minutes
```

Observed: In this part of the code, my first run takes me almost an hour but interrupted by crash, the last attempt went smoothly and takes no longer 5 minutes, the accuracy still went up a little by 1%.

In [10]:

```
import time
start_time = time.time()
import sys
from matplotlib import pyplot
from tensorflow.keras.datasets import fashion_mnist
from keras.models import load_model
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.optimizers import SGD
from keras.layers import Dropout
```

In [11]:

```
def LoadDataset():
  (trainX, trainY), (testX, testY) = fashion_mnist.load_data()
  trainY= to_categorical(trainY)
  testY = to_categorical(testY)
  trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))
  testX = testX.reshape((testX.shape[0], 28, 28, 1))
  return trainX, trainY, testX, testY

def PrepPixels(Train, Test):
  train_norm = Train.astype('float32')
  test_norm = Test.astype('float32')
  train_norm = train_norm / 255.0
```

```
  test_norm = test_norm / 255.0
  return train_norm, test_norm
```

In [12]:

```
def RunTestHarness2():
  (trainX, trainY, testX, testY) = LoadDataset()
  trainX, testX = PrepPixels(trainX, testX)
  model = CNNModel()
  model = load_model('/content/drive/MyDrive/CPE 019 (Retake)/Assignment 9.1/final_model.h5')
  _, acc = model.evaluate(testX, testY, verbose=0)
  print('> %.3f' % (acc * 100.0))
```

In [13]:

```
RunTestHarness2()
end_time = time.time()

total_time = end_time - start_time
print("Total time taken:", total_time/60, "minutes")
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use
`learning_rate` or use the legacy optimizer,
e.g.,tf.keras.optimizers.legacy.SGD.
```

```
> 92.250
Total time taken: 0.11020640134811402 minutes
```

In [4]:

```
import numpy as np
from keras.models import load_model
from tensorflow.keras.datasets import fashion_mnist
from keras.utils import to_categorical

def load_image(index):
    (X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()
    image = X_test[index]
    image = image.reshape(1, 28, 28, 1)
    image = image.astype('float32') / 255.0
    return image, y_test[index]

def RunExample(image_index):
  img, true_label = load_image(image_index)
  model = load_model('/content/drive/MyDrive/CPE 019 (Retake)/Assignment 9.1/final_model.h5')
  FinalResult = np.argmax(model.predict(img))
  print("Predicted Label:", FinalResult)
  print("Actual Label:", true_label)
```

In [6]:

```
RunExample(6)
```

```
1/1 [==============================] - 0s 105ms/step
Predicted Label: 4
Actual Label: 4
```

Observation: As seen in the final result, the predication is correct to the actual label, with the accuracy of 92.25%

Conclusion¶

- In this activity, I were able to implement Conulutionary Neural Network, although executing this activity takes long than I expected. Perfroming a 3D model is such a hassle at the same time a great experience while performing. Implementing Convolutionary Neural Network will be great use to me in the future experiment or project.