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 |
|  |  |
| \*\*Hands-on Activity 9.1\*\* | \*\*Convolutional Neural Network\*\* |
| **Name** | Calvadores, Kelly Joseph |
| **Section** | CPE32S1 |
| **Date Performed**: | July 07, 2024 |
| **Date Submitted**: | July , 2024 |
| **Instructor**: | Engr. Roman M. Richard |

# Choose any dataset applicable to an image classification problem[¶](#X5a66fa3b21f3568425981e314e691a059adf65b)

# Explain your datasets and the problem being addressed.[¶](#X328b202143c2b9a66105f89c123e69f19d2bc28)

* 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:[¶](#Xa9e61c60a744a2754d934131434dcc5e041e97d)

## Using your dataset, create a baseline model of the CNN[¶](#X66f84bc34d777ce43686fac3b64e735a5df3e34)

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

![](data:image/png;base64;base64,)

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

![](data:image/png;base64;base64,)

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

![](data:image/png;base64;base64,)

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

![](data:image/png;base64;base64,)

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)

![](data:image/png;base64;base64,)

**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[¶](#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()

![](data:image/png;base64;base64,)

## Perform feature standardization[¶](#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).

![](data:image/png;base64;base64,)

**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

![](data:image/png;base64;base64,)

## Perform ZCA whitening of your images[¶](#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

![](data:image/png;base64;base64,)

**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')

![](data:image/png;base64;base64,)

![](data:image/png;base64;base64,)

**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[¶](#Xdec256d0bbc3ecf6efc122df282ac72caa5a1cf)

### Random Rotation[¶](#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

![](data:image/png;base64;base64,)

**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[¶](#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

![](data:image/png;base64;base64,)

**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[¶](#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

![](data:image/png;base64;base64,)

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

## Save augmented image data to disk[¶](#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

![](data:image/png;base64;base64,)

## Develop a test harness to develop a robust evaluation of a model and establish a baseline of performance for a classification task[¶](#X3c1b16e0d8b4b917cf149f3a324f9e3a48d31cb)

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,)

![](data:image/png;base64;base64,)

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[ ]:

![](data:image/png;base64;base64,)

**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.[¶](#X2adbf67b9601792e523e0169027e85e4dd111e4)

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[ ]:

![](data:image/png;base64;base64,)

**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.[¶](#X40ab90620e802fc03b02de194df7ae77bbeb9c2)

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 92.250%

# Conclusion[¶](#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.

In [ ]: