# Assignment 9.1 : Convolutional Neural Network

| **Technological Institute of the Philippines** | **Quezon City - Computer Engineering** |
|---|---|
| Course Code: | CPE 019 |
| Code Title: | Emerging Technologies in CpE 2 - Fundamentals of Data Science |
| 1st Semester | AY 2023-2024 |

| | |
|---|---|
| **Assignment 9.1 ** | **Convolutional Neural Networks** |
| **Name** | William Laurence M. Ramos |
| **Section** | CPE32S1 |
| **Date Performed**: | 08/07/2024 |
| **Date Submitted**: | 10/07/2024 |
| **Instructor**: | Engr. Roman M. Richard |

## Instructions:

1. Choose any dataset applicable to an image classification problem
2. Explain your datasets and the problem being addressed.
3. Show evidence that you can do the following:
    1. Using your dataset, create a baseline model of the CNN
    2. Perform image augmentation
    3. Perform feature standardization
    4. Perform ZCA whitening of your images
    5. Augment data with random rotations, shifts, and flips
    6. Save augmented image data to disk
    7. Develop a test harness to develop a robust evaluation of a model and establish a baseline of performance for a classification task
    8. Explore extensions to a baseline model to improve learning and model capacity.
    9. Develop a finalized model, evaluate the performance of the final model, and use it to make predictions on new images.
4. Submit the link to your Google Colab (make sure that it is accessible to me)

# Image Classification Problem

# FASHION-MNIST Image Classification Dataset

A popular dataset in machine learning research and education, especially in computer vision, is Fashion-MNIST. It acts as a standard against which to evaluate and contrast various machine learning models and techniques.

A selection of 28x28 grayscale pictures of apparel and accessories, including shirts, dresses, shoes, and purses, are included in the dataset. Every image has a label attached to it that describes the kind of garment it depicts. In total, there are ten groups or divisions, each of which represents a distinct kind of apparel.

Classification is the main issue that Fashion-MNIST aims to solve. The objective is to create machine learning models that can correctly estimate the item's class label given an image of the clothes.

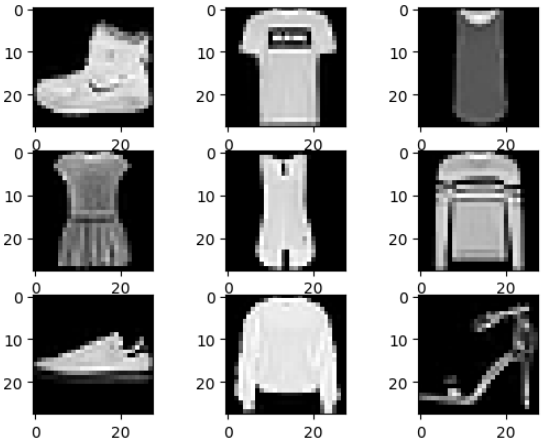
## There are 10 classes in the dataset which is assigned to the following labels:

- 0: T-shirt/top
- 1: Trouser
- 2: Pullover
- 3: Dress
- 4: Coat
- 5: Sandal
- 6: Shirt
- 7: Sneaker
- 8: Bag
- 9: Ankle boot

## Load the dataset

```
# Loading the FASHION-MNIST dataset
from tensorflow.keras.datasets import fashion_mnist
from matplotlib import pyplot as plt

# load dataset
(trainX, trainy), (testX, testy) = fashion_mnist.load_data()

# summarize loaded dataset
print('Train: X=%s, y=%s' % (trainX.shape, trainy.shape))
print('Test: X=%s, y=%s' % (testX.shape, testy.shape))

# plot first few images
for i in range(9):
    # define subplot
    plt.subplot(330 + 1 + i)
    # plot raw pixel data
    plt.imshow(trainX[i], cmap=plt.get_cmap('gray'))

# show the figure
plt.show()
```

```
Train: X=(60000, 28, 28), y=(60000,)
Test: X=(10000, 28, 28), y=(10000,)
```



Here is a sample of images from our Fashion-MNIST dataset.

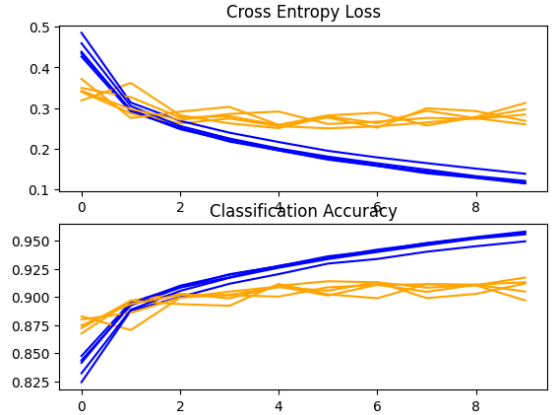
## Task 1: Using your dataset, create a baseline model of the CNN

```
import time
start_time = time.time()

# baseline cnn model for FASHION-MNIST
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 fashion_mnist
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


# load train and test dataset
def load_dataset():
    # load dataset
    (trainX, trainY), (testX, testY) = fashion_mnist.load_data()
    # reshape dataset to have a single channel
    trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))
    testX = testX.reshape((testX.shape[0], 28, 28, 1))
    # one hot encode target values
    trainY = to_categorical(trainY)
    testY = to_categorical(testY)
    return trainX, trainY, testX, testY

# scale pixels
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
```

```
# define cnn model
def define_model():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', input
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
    model.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dense(10, activation='softmax'))
    # compile model
    opt = SGD(learning_rate=0.01, momentum=0.9)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
    return model

# evaluate a model using k-fold cross-validation
def evaluate_model(dataX, dataY, n_folds=5):
    scores, histories = list(), list()
    # prepare cross validation
    kfold = KFold(n_folds, shuffle=True, random_state=1)
    # enumerate splits
    for train_ix, test_ix in kfold.split(dataX):
        # define model
        model = define_model()
        # select rows for train and test
        trainX, trainY, testX, testY = dataX[train_ix], dataY[train_ix], dataX[test_ix], d
        # fit model
        history = model.fit(trainX, trainY, epochs=10, batch_size=32, validation_data=(tes
        # evaluate model
        _, acc = model.evaluate(testX, testY, verbose=0)
        print('> %.3f' % (acc * 100.0))
        # stores scores
        scores.append(acc)
        histories.append(history)
    return scores, histories
```

```
    # plot diagnostic learning curves
    def summarize_diagnostics(histories):
        for i in range(len(histories)):
            # plot loss
            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')
            # plot accuracy
            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()

    # summarize model performance
    def summarize_performance(scores):
        # print summary
        print('Accuracy: mean=%.3f std=%.3f, n=%d' % (mean(scores)*100, std(scores)*100, l
        # box and whisker plots of results
        plt.boxplot(scores)
        plt.show()


# run the test harness for evaluating a model
def run_test_harness():
    # load dataset
    trainX, trainY, testX, testY = load_dataset()
    # prepare pixel dat
    trainX, testX = prep_pixels(trainX, testX)
    # evaluate model
    scores, histories = evaluate_model(trainX, trainY)
    # learning curves
    summarize_diagnostics(histories)
    # summarize estimated performance
    summarize_performance(scores)

# entry point, run the test harness
run_test_harness()

# Total time
end_time = time.time()

total_time = end_time - start_time
print("Total time taken in seconds:", total_time, "seconds")
```

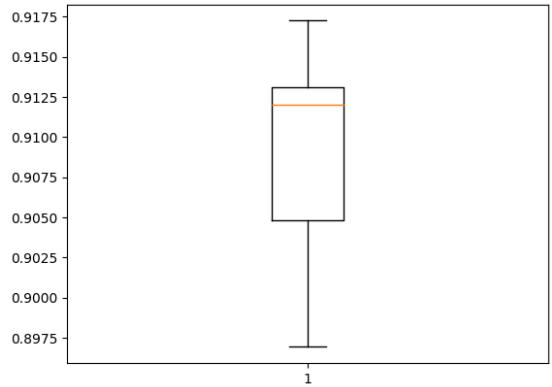
```
> 90.483
> 91.725
> 91.200
> 89.692
> 91.308
```







```
Accuracy: mean=90.882 std=0.717, n=5
```




```
Total time taken in seconds: 362.57258224487305 seconds
```

```
print("Total time taken in minutes:", total_time/60, "minutes")
```

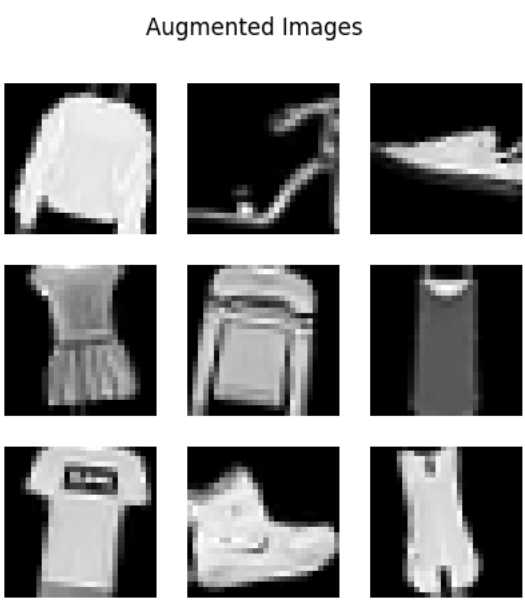
```
Total time taken in minutes: 6.0428763707478845 minutes
```

For every fold during training, the learning curves (accuracy and loss) are plotted using the summarize_diagnostics() function. This aids in recognising any overfitting or underfitting problems and visually representing the model's training progress. A summary of the model's performance based on the accuracy scores acquired from cross-validation is printed by the summarize_performance() function. Additionally, a box plot is used to display the accuracy score distribution. Everything is coordinated by the run_test_harness() function, which loads the data, assesses the model, and summarises its output. It acts as the point of entry for carrying out the experiment.

## Task 2: Perform image augmentation

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

# Load CIFAR-10 dataset
(trainX, trainy), (testX, testy) = fashion_mnist.load_data()

# Create an ImageDataGenerator for augmentation
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True
)

# Plot original images
plt.figure(figsize=(5, 5))
for i in range(9):
    plt.subplot(330 + 1 + i)
    plt.imshow(trainX[i])
    plt.axis('off')
plt.suptitle('Original Images')
plt.show()
```


```
# Reshape the input data to include a single channel
import numpy as np
trainX = np.expand_dims(trainX, axis=-1)

# Generate augmented images
it = datagen.flow(trainX[:9], batch_size=9)
augmented_images = it.next()

# Plot augmented images
plt.figure(figsize=(5, 5))
for i in range(9):
    plt.subplot(330 + 1 + i)
    plt.imshow(augmented_images[i].astype('uint8'), cmap='gray')
    plt.axis('off')
plt.suptitle('Augmented Images')
plt.show()
```



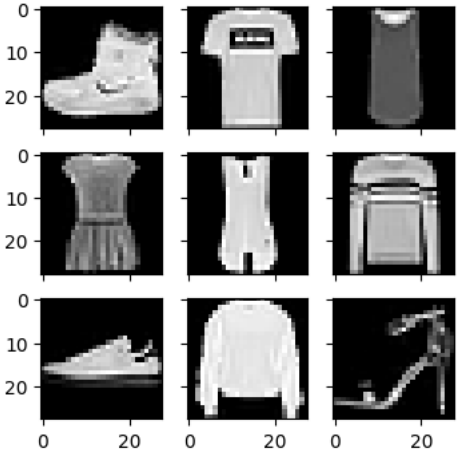

We used image augmentation for this job, comparing the results to the original photos to observe how our output changed. It is evident that it includes image augmentation techniques including shifts, flips, and random rotations. Since we will be performing our final picture augmentation in Task 4-5, we only utilised this code as an example of how to implement image augmentation.

Double-click (or enter) to edit

## Task 3: Perform feature standardization

```
# Standardize images across the dataset, mean=0, stdev=1
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt

# load data
(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()

# reshape to be [samples][width][height][channels]
X_train = X_train.reshape((X_train.shape[0], 28, 28, 1))
X_test = X_test.reshape((X_test.shape[0], 28, 28, 1))

# convert from int to float
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

# define data preparation
datagen = ImageDataGenerator(featurewise_center=True, featurewise_std_normalization=True)

# fit parameters from data
datagen.fit(X_train)

# configure batch size and retrieve one batch of images
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())

    # create a grid of 3x3 images
    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"))

    # show the plot
    plt.show()
    break
```

```
-0.8102582 0.0542206 2.0224075
```



The values you see above the picture represent the batch's minimum, mean, and maximum pixel intensities after they have been standardised.

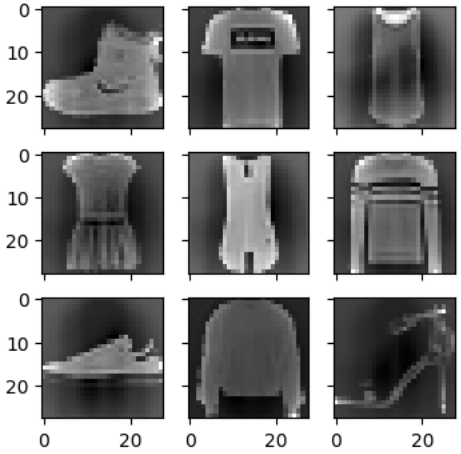
**Minimum: -0.8102582**

**Mean: 0.0542206**

**Maximum: 2.0224075**

```
# Standardize images across the dataset, every pixel has mean=0, stdev=1
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt

# load data
(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()

# reshape to be [samples][width][height][channels]
X_train = X_train.reshape((X_train.shape[0], 28, 28, 1))
X_test = X_test.reshape((X_test.shape[0], 28, 28, 1))

# convert from int to float
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

# define data preparation
datagen = ImageDataGenerator(featurewise_center=True, featurewise_std_normalization=True)

# fit parameters from data
datagen.mean = X_train.mean(axis=0)
datagen.std = X_train.std(axis=0)

# configure batch size and retrieve one batch of images
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())

    # create a grid of 3x3 images
    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"))

    # show the plot
    plt.show()
    break
```

```
-2.4007795 0.056358527 7.556089
```




**Minimum: -2.4007795**

**Mean: 0.056358527**

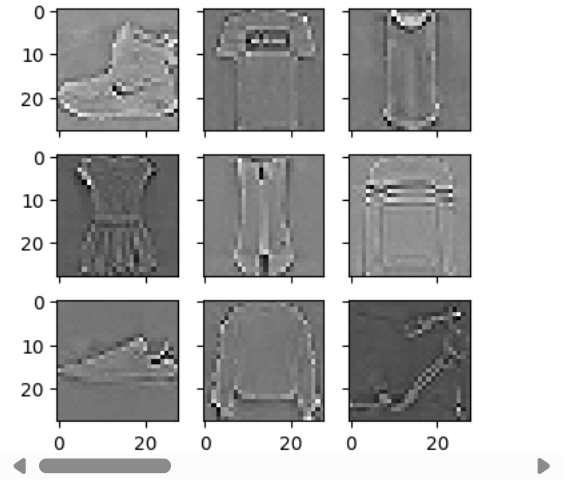
**Maximum: 7.556089**

Here, we use our own algorithm in place of the fit() function. As we can see, the maximum values have increased significantly. Our range of values has expanded.

## Task 4: Perform ZCA whitening of your images

```
# ZCA Whitening
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt

# load data
(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()

# reshape to be [samples][width][height][channels]
X_train = X_train.reshape((X_train.shape[0], 28, 28, 1))
X_test = X_test.reshape((X_test.shape[0], 28, 28, 1))

# convert from int to float
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

# define data preparation
datagen = ImageDataGenerator(featurewise_center=True, featurewise_std_normalization=True,

# fit parameters from data
X_mean = X_train.mean(axis=0)
datagen.fit(X_train - X_mean)

# configure batch size and retrieve one batch of images
for X_batch, y_batch in datagen.flow(X_train - X_mean, y_train, batch_size=9, shuffle=Fals
    print(X_batch.min(), X_batch.mean(), X_batch.max())

    # create a grid of 3x3 images
    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='gray')

    # show the plot
    plt.show()
    break
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/prep
  warnings.warn(
-7.6035457 0.0044841086 13.209011
```

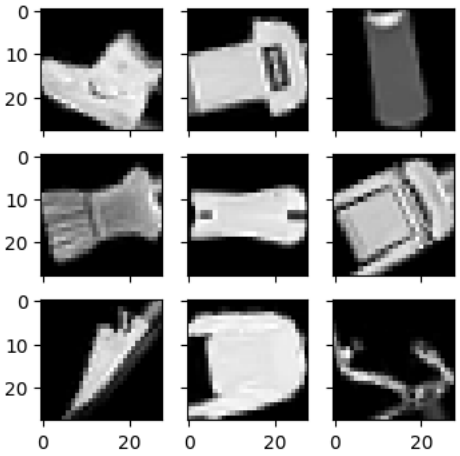



ZCA whitening can enhance the effectiveness of our machine learning algorithms by lowering correlations and redundancies among pixel values in our image data. I can see that each image's outline has been highlighted.

## Task 5: Augment data with random rotations, shifts, and flips

### Random Rotations

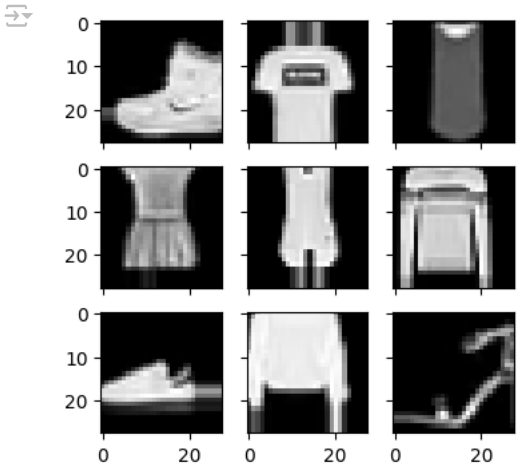
```
# Random Rotations
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt

# load data
(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()

# reshape to be [samples][width][height][channels]
X_train = X_train.reshape((X_train.shape[0], 28, 28, 1))
X_test = X_test.reshape((X_test.shape[0], 28, 28, 1))

# convert from int to float
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

# define data preparation
datagen = ImageDataGenerator(rotation_range=90)

# configure batch size and retrieve one batch of images
for X_batch, y_batch in datagen.flow(X_train, y_train, batch_size=9, shuffle=False):
    # create a grid of 3x3 images
    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(28,28), cmap=plt.get_cmap("gray"))

    # show the plot
    plt.show()
    break
```




Improving the model's capacity to adapt to changes in object orientation requires adding random rotations to the dataset during augmentation. The model learns to identify objects regardless of their orientation by being trained with randomly rotated images, which improves performance when faced with undiscovered orientations during inference.

## Shifts

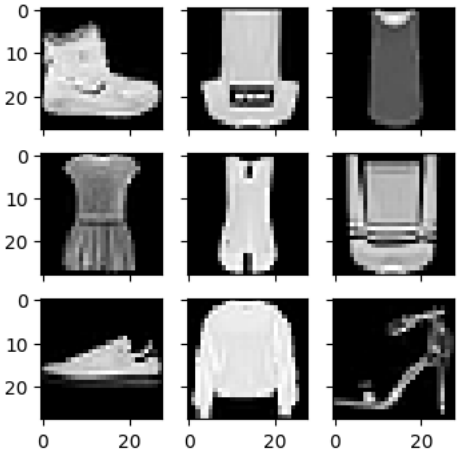
```
# Random Shifts
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt

# load data
(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()

# reshape to be [samples][width][height][channels]
X_train = X_train.reshape((X_train.shape[0], 28, 28, 1))
X_test = X_test.reshape((X_test.shape[0], 28, 28, 1))

# convert from int to float
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

# define data preparation
shift = 0.2
datagen = ImageDataGenerator(width_shift_range=shift, height_shift_range=shift)

# configure batch size and retrieve one batch of images
for X_batch, y_batch in datagen.flow(X_train, y_train, batch_size=9, shuffle=False):
    # create a grid of 3x3 images
    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(28,28), cmap=plt.get_cmap("gray"))

    # show the plot
    plt.show()
    break
```



In order to train the model to be invariant to changes in object position within the image, data augmentation with random shifts is essential. In real life, things could not always be precisely centred or aligned inside pictures; instead, they might appear at varied locations as a result of changes in object placement or camera framing.

## Flips

```
# Random Flips
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt

# load data
(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()

# reshape to be [samples][width][height][channels]
X_train = X_train.reshape((X_train.shape[0], 28, 28, 1))
X_test = X_test.reshape((X_test.shape[0], 28, 28, 1))

# convert from int to float
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

# define data preparation
datagen = ImageDataGenerator(horizontal_flip=True, vertical_flip=True)

# configure batch size and retrieve one batch of images
for X_batch, y_batch in datagen.flow(X_train, y_train, batch_size=9, shuffle=False):
    # create a grid of 3x3 images
    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(28,28), cmap=plt.get_cmap("gray"))

    # show the plot
    plt.show()
    break
```




It is imperative to augment the dataset with random flips, like horizontal or vertical flips, to improve the model's recognition performance of objects from any orientation or point of view. The model learns to detect object properties that are invariant to such transformations by being trained using randomly flipped photos.

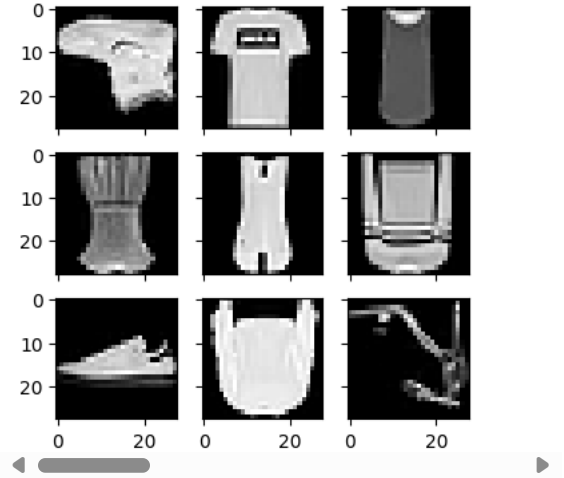
## Task 6: Save augmented image data to disk

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# Save augmented images to file
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt

# load data
(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()

# reshape to be [samples][width][height][channels]
X_train = X_train.reshape((X_train.shape[0], 28, 28, 1))
X_test = X_test.reshape((X_test.shape[0], 28, 28, 1))

# convert from int to float
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

# define data preparation
datagen = ImageDataGenerator(horizontal_flip=True, vertical_flip=True, zca_whitening=True)

# configure batch size and retrieve one batch of images
for X_batch, y_batch in datagen.flow(X_train, y_train, batch_size=9, shuffle=False,
                                     save_to_dir='/content/drive/MyDrive/Assignment9.1', s
    # create a grid of 3x3 images
    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(28,28), cmap=plt.get_cmap("gray"))
    # show the plot
    plt.show()
    break
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/prep
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/prep
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/prep
  warnings.warn(
```



The augmented image data is now saved in our Google Drive folder 'Assignment 9.1'. The saved augmented image data is shown below:

```
from google.colab.patches import cv2_imshow
import cv2

image_paths = ['/content/drive/MyDrive/Assignment9.1/aug_0_4108.png',
               '/content/drive/MyDrive/Assignment9.1/aug_1_1218.png',
               '/content/drive/MyDrive/Assignment9.1/aug_2_4919.png',
               '/content/drive/MyDrive/Assignment9.1/aug_3_5867.png',
               '/content/drive/MyDrive/Assignment9.1/aug_4_1344.png',
               '/content/drive/MyDrive/Assignment9.1/aug_5_2852.png',
               '/content/drive/MyDrive/Assignment9.1/aug_6_2246.png',
               '/content/drive/MyDrive/Assignment9.1/aug_7_6635.png',
               '/content/drive/MyDrive/Assignment9.1/aug_8_7816.png']

# Display each image
for image_path in image_paths:
    # Load and display the image
    img = cv2.imread(image_path)
    cv2_imshow(img)
```

## Task 7: Develop a test harness to develop a robust evaluation of a model and establish a baseline of performance for a classification task

We create a test harness and set up three baselines with varying categorization performance for this assignment.

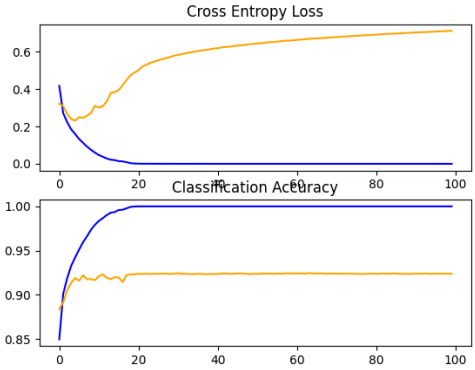
### Baseline 1

```
import time
start_time = time.time()
# test harness for evaluating models on the Fashion_mnist dataset
import sys
from matplotlib import pyplot
from 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

# load train and test dataset
def load_dataset():
    # load dataset
    (trainX, trainY), (testX, testY) = fashion_mnist.load_data()
    # one hot encode target values
    trainY = to_categorical(trainY)
    testY = to_categorical(testY)
    return trainX, trainY, testX, testY

# scale pixels
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

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

# plot diagnostic learning curves
def summarize_diagnostics(history):
    # plot loss
    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')
    # plot accuracy
```

```
    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')
    # save plot to file
    filename = '/content/drive/MyDrive/Assignment9.1/plot1.png'
    pyplot.savefig(filename)
    pyplot.close()

# run the test harness for evaluating a model
def run_test_harness():
    # load dataset
    trainX, trainY, testX, testY = load_dataset()
    # prepare pixel data
    trainX, testX = prep_pixels(trainX, testX)
    # define model
    model = define_model()
    # fit model
    history = model.fit(trainX, trainY, epochs=100, batch_size=64, validation_data=(testX,
    # evaluate model
    _, acc = model.evaluate(testX, testY, verbose=0)
    print('> %.3f' % (acc * 100.0))
    # learning curves
    summarize_diagnostics(history)

# entry point, run the test harness
run_test_harness()

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
> 92.370
Total time taken: 7.411065371831258 minutes
```

```
import cv2
from google.colab.patches import cv2_imshow

img = cv2.imread('/content/drive/MyDrive/Assignment9.1/plot1.png')
cv2_imshow(img)
```


Cross Entropy Loss
0.6
0.4
0.2
0.0
0
20
40
60
80
100
Classification Accuracy
1.00
0.95
0.90
0.85
0
20
40
60
80
100

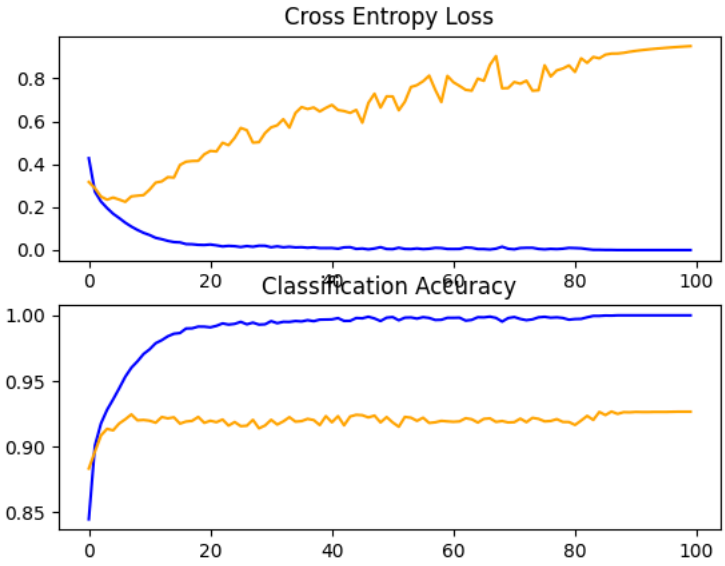
## Baseline 2

```
import time
start_time = time.time()
# test harness for evaluating models on the Fashion_mnist dataset
import sys
from matplotlib import pyplot
from 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

# load train and test dataset
def load_dataset():
    # load dataset
    (trainX, trainY), (testX, testY) = fashion_mnist.load_data()
    # one hot encode target values
    trainY = to_categorical(trainY)
    testY = to_categorical(testY)
    return trainX, trainY, testX, testY

# scale pixels
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

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

# plot diagnostic learning curves
def summarize_diagnostics(history):
    # plot loss
    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')
    # plot accuracy
    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')
    # save plot to file
    filename = '/content/drive/MyDrive/Assignment9.1/plot2.png'
    pyplot.savefig(filename)
    pyplot.close()

# run the test harness for evaluating a model
def run_test_harness():
    # load dataset
    trainX, trainY, testX, testY = load_dataset()
    # prepare pixel data
    trainX, testX = prep_pixels(trainX, testX)
    # define model
    model = define_model()
    # fit model
    history = model.fit(trainX, trainY, epochs=100, batch_size=64, validation_data=(testX,
    # evaluate model
    _, acc = model.evaluate(testX, testY, verbose=0)
    print('> %.3f' % (acc * 100.0))
    # learning curves
    summarize_diagnostics(history)

# entry point, run the test harness
run_test_harness()

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
> 92.670
Total time taken: 8.41004524230957 minutes
```

```
import cv2
from google.colab.patches import cv2_imshow

img = cv2.imread('/content/drive/MyDrive/Assignment9.1/plot2.png')
cv2_imshow(img)
```


Cross Entropy Loss
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
Classification Accuracy
1.00
0.95
0.90
0.85
0
20
40
60
80
100

Baseline 3

```
import time
start_time = time.time()
# test harness for evaluating models on the Fashion_mnist dataset
import sys
from matplotlib import pyplot
from 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

# load train and test dataset
def load_dataset():
    # load dataset
    (trainX, trainY), (testX, testY) = fashion_mnist.load_data()
    # one hot encode target values
    trainY = to_categorical(trainY)
    testY = to_categorical(testY)
    return trainX, trainY, testX, testY

# scale pixels
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

# define cnn model
def define_model():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', paddi
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', paddi
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', paddi
    model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', paddi
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padd
    model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padd
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
    model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dense(10, activation='softmax'))
    # compile model
    opt = SGD(lr=0.001, momentum=0.9)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
    return model

# plot diagnostic learning curves
def summarize_diagnostics(history):
```

```
    # plot loss
    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')
    # plot accuracy
    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')
    # save plot to file
    filename = '/content/drive/MyDrive/Assignment9.1/plot3.png'
    pyplot.savefig(filename)
    pyplot.close()

# run the test harness for evaluating a model
def run_test_harness():
    # load dataset
    trainX, trainY, testX, testY = load_dataset()
    # prepare pixel data
    trainX, testX = prep_pixels(trainX, testX)
    # define model
    model = define_model()
    # fit model
    history = model.fit(trainX, trainY, epochs=100, batch_size=64, validation_data=(testX,
    # evaluate model
    _, acc = model.evaluate(testX, testY, verbose=0)
    print('> %.3f' % (acc * 100.0))
    # learning curves
    summarize_diagnostics(history)

# entry point, run the test harness
run_test_harness()

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
> 91.710
Total time taken: 10.435135853290557 minutes
```

```
import cv2
from google.colab.patches import cv2_imshow

img = cv2.imread('/content/drive/MyDrive/Assignment9.1/plot3.png')
cv2_imshow(img)
```


**The highest among the baseline is 92.670 which comes from our second testing of harnesss.**

## Task 8: Explore extensions to a baseline model to improve learning and model capacity.

I employed dropout regularisation to raise our baseline even further. A straightforward method called "dropout" causes nodes to be randomly removed from the network. Because the surviving nodes have to adjust to take up the slack left by the eliminated nodes, it has a regularising effect.

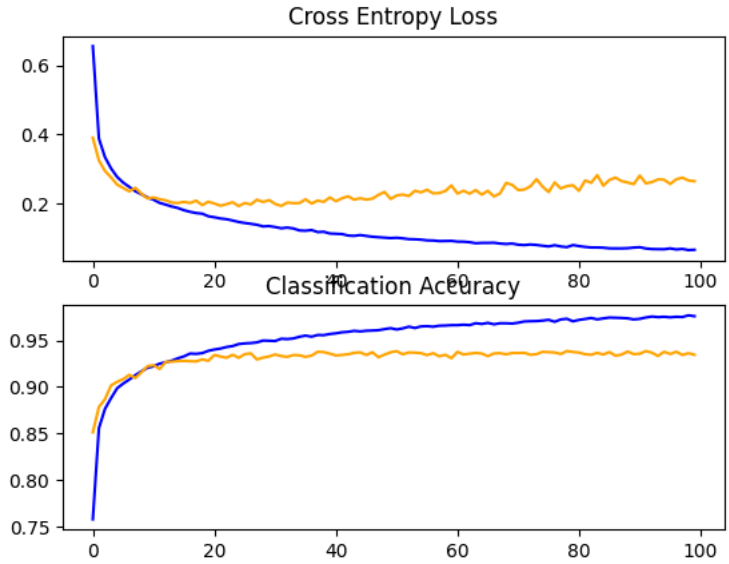
```
import time
start_time = time.time()
# baseline model with dropout on the Fashion-Mnist dataset
import sys
from matplotlib import pyplot
from keras.datasets import fashion_mnist
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Dropout
from keras.optimizers import SGD

# load train and test dataset
```

```
# load train and test dataset
def load_dataset():
  # load dataset
  (trainX, trainY), (testX, testY) = fashion_mnist.load_data()
  # one hot encode target values
  trainY = to_categorical(trainY)
  testY = to_categorical(testY)
  return trainX, trainY, testX, testY

# scale pixels
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

# define cnn model
def define_model():
  model = Sequential()
  model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding=
  model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding=
  model.add(MaxPooling2D((2, 2)))
  model.add(Dropout(0.2))
  model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding=
  model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding=
  model.add(MaxPooling2D((2, 2)))
  model.add(Dropout(0.2))
  model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding
  model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding
  model.add(MaxPooling2D((2, 2)))
  model.add(Dropout(0.2))
  model.add(Flatten())
  model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
  model.add(Dropout(0.2))
  model.add(Dense(10, activation='softmax'))
  # compile model
  opt = SGD(lr=0.001, momentum=0.9)
  model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
  return model

# plot diagnostic learning curves
def summarize_diagnostics(history):
  # plot loss
  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')
  # plot accuracy
  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')
  # save plot to file
```

```
  filename = '/content/drive/MyDrive/Assignment9.1/plot4.png'
  pyplot.savefig(filename)
  pyplot.close()

# run the test harness for evaluating a model
def run_test_harness():
  # load dataset
  trainX, trainY, testX, testY = load_dataset()
  # prepare pixel data
  trainX, testX = prep_pixels(trainX, testX)
  # define model
  model = define_model()
  # fit model
  history = model.fit(trainX, trainY, epochs=100, batch_size=64, validation_data=(testX, te
  # evaluate model
  _, acc = model.evaluate(testX, testY, verbose=0)
  print('> %.3f' % (acc * 100.0))
  # learning curves
  summarize_diagnostics(history)

# entry point, run the test harness
run_test_harness()

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
> 93.450
Total time taken: 12.424622444311778 minutes
```

```
import cv2
from google.colab.patches import cv2_imshow

img = cv2.imread('/content/drive/MyDrive/Assignment9.1/plot4.png')
cv2_imshow(img)
```




From what I can see, 92.670 is our model's baseline. Following the application of the dropout regularisation technique, our accuracy increases to 93.450 from 92.670. The accuracy is also far greater and more consistent than our baseline, as the dropout regularisation graph demonstrates.

## Task 9: Develop a finalized model, evaluate the performance of the final model, and use it to make predictions on new images.

### Saving a Finalized Model

```
import time
start_time = time.time()
# Develop a finalized model with Fashion-Mnist dataset
import sys
from matplotlib import pyplot
from 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.layers import Dropout
from keras.optimizers import SGD

# load train and test dataset
def load_dataset():
    # load dataset
    (trainX, trainY), (testX, testY) = fashion_mnist.load_data()
    # one hot encode target values
    trainY = to_categorical(trainY)
    testY = to_categorical(testY)
    return trainX, trainY, testX, testY

# scale pixels
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

# define cnn model
def define_model():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', paddi
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', paddi
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.2))
    model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', paddi
    model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', paddi
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.2))
    model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padd
    model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padd
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.2))
    model.add(Flatten())
    model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dropout(0.2))
    model.add(Dense(10, activation='softmax'))
    # compile model
    opt = SGD(lr=0.001, momentum=0.9)
```

```
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
    return model

# run the test harness for evaluating a model
def run_test_harness():
    # load dataset
    trainX, trainY, testX, testY = load_dataset()
    # prepare pixel data
    trainX, testX = prep_pixels(trainX, testX)
    # define model
    model = define_model()
    # fit model
    model.fit(trainX, trainY, epochs=100, batch_size=64, validation_data=(testX, testY), v
    # save model
    model.save('/content/drive/MyDrive/Assignment9.1/final_model.h5')

# entry point, run the test harness
run_test_harness()

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
Total time taken: 12.410495821634928 minutes
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning
  saving_api.save_model(
```

Our finished model was saved in the.h5 format in this code. It will be saved to our Google Drive in the Assignment9.1 directory.

## Loading and Evalutation of Finalized Model

```
import time
start_time = time.time()

# Evaluate the deep model on the test dataset
from keras.datasets import fashion_mnist
from keras.models import load_model

# load train and test dataset
def load_dataset():
    # load dataset
    (trainX, trainY), (testX, testY) = fashion_mnist.load_data()
    # one hot encode target values
    trainY = to_categorical(trainY)
    testY = to_categorical(testY)
    return trainX, trainY, testX, testY

# scale pixels
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

# run the test harness for evaluating a model
def run_test_harness():
    # load dataset
    trainX, trainY, testX, testY = load_dataset()
    # prepare pixel data
    trainX, testX = prep_pixels(trainX, testX)
    # define model
    model = load_model('/content/drive/MyDrive/Assignment9.1/final_model.h5')
    # Evaluate model on test dataset
    _, acc = model.evaluate(testX, testY, verbose=0)
    print('> %.3f' % (acc * 100.0))

# entry point, run the test harness
run_test_harness()

end_time = time.time()

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

```
> 93.080
Total time taken: 0.04472893476486206 minutes
```

We can now use the load_model function to assess and use the final model after saving it.

## Predicting images using the Finalized Model

There are 10 classes in the dataset which is assigned to the following labels:

- 0: T-shirt/top
- 1: Trouser
- 2: Pullover
- 3: Dress
- 4: Coat
- 5: Sandal
- 6: Shirt
- 7: Sneaker
- 8: Bag
- 9: Ankle boot

We need a sample image for us to use and predict using our finalized model. You can see from below is an image extracted from the Fashion-Mnist test dataset.

```
import matplotlib.pyplot as plt
from PIL import Image

# Load the image
image_path = '/content/drive/MyDrive/Assignment9.1/10010.png'
img = Image.open(image_path)

# Display the image with zoom
plt.figure(figsize=(5, 5))  # Adjust the figsize to control the zoom level
plt.imshow(img)
plt.axis('off')
plt.show()
```