**CSE4132: Artificial Intelligence Lab Assignment\_18-12-2024**

Nusrat Jannat

Roll: 2012176145

21 December,2024

**Ques:1** Build a fully connected neural network (FCNN) and a convolutional neural

network (CNN) for classifying 10 classes of images.

**Ans:1**

FCNN for classifying 10 classes of images:

import tensorflow as tf

from tensorflow.keras.models import Model

from tensorflow.keras.layers import Input, Dense, Flatten

from tensorflow.keras.utils import to\_categorical

num\_classes = 10

inputs = Input((28,28, 1))

x = Flatten()(inputs)

x = Dense(512, activation='relu')(x)

x = Dense(256, activation='relu')(x)

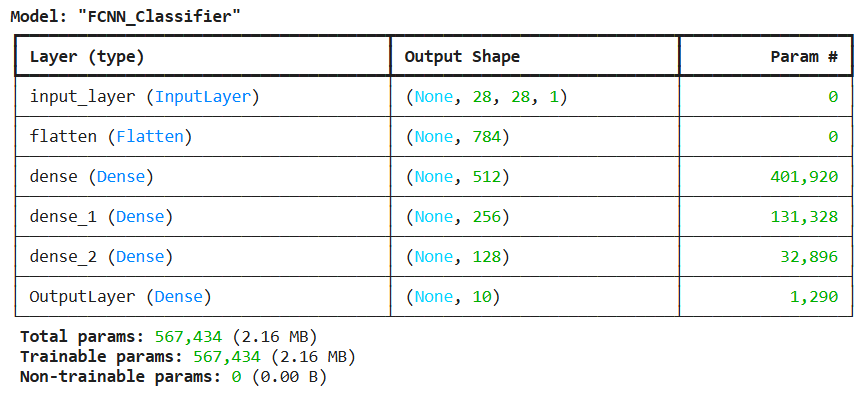
x = Dense(128, activation='relu')(x)

outputs = Dense(num\_classes, activation='softmax', name="OutputLayer")(x)

model = Model(inputs, outputs, name="FCNN\_Classifier")

model.summary()

Output:



CNN for classifying 10 classes of images:

import tensorflow as tf

from tensorflow.keras.models import Model

from tensorflow.keras.layers import Input, Conv2D, Flatten, Dense

num\_classes = 10

inputs = Input((28,28, 1))

x = Conv2D(32, kernel\_size=(3, 3), padding = 'same', activation='relu')(inputs) # 32 filters

x = Conv2D(64, kernel\_size=(3, 3), padding = 'same', activation='relu')(x) # 64 filters

x = Conv2D(128, kernel\_size=(3, 3), padding = 'same', activation='relu')(x) # 128 filters

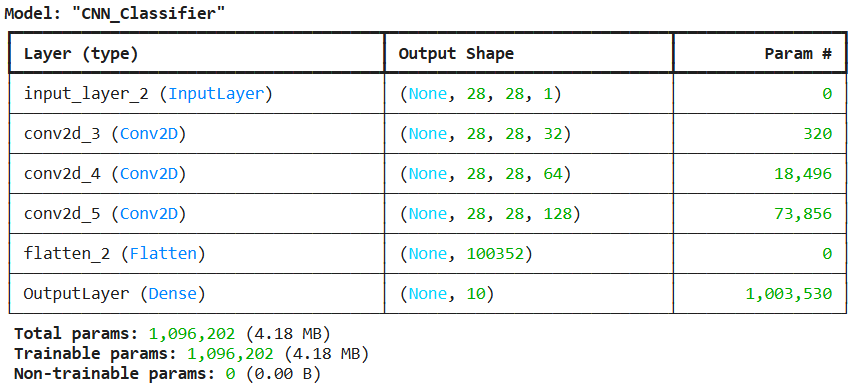
x = Flatten()(x)

outputs = Dense(num\_classes, activation='softmax', name="OutputLayer")(x)

model = Model(inputs, outputs, name="CNN\_Classifier")

model.summary()

Output:



**Ques:2** Train and test your FCNN and CNN by the Fashion dataset. Discuss your

results by comparing performance between two types of networks.

**Ans:2**

Train and test FCNN:

from tensorflow.keras.datasets import fashion\_mnist

import matplotlib.pyplot as plt

import numpy as np

from tensorflow.keras.utils import to\_categorical

from tensorflow.keras.layers import Input, Flatten, Dense

from tensorflow.keras.models import Model

# Helper function to display images

def display\_img(img\_set, title\_set):

n = len(title\_set)

for i in range(n):

plt.subplot(3, 3, i + 1)

plt.imshow(img\_set[i], cmap='gray')

plt.title(title\_set[i])

plt.show()

plt.close()

# Load the Fashion-MNIST dataset

(trainX, trainY), (testX, testY) = fashion\_mnist.load\_data()

# Investigate loaded data

print('trainX.shape: {}, trainY.shape: {}, testX.shape: {}, testY.shape: {}'.format(trainX.shape, trainY.shape, testX.shape, testY.shape))

print('trainX.dtype: {}, trainY.dtype: {}, testX.dtype: {}, testY.dtype: {}'.format(trainX.dtype, trainY.dtype, testX.dtype, testY.dtype))

print('trainX.Range: {} - {}, testX.Range: {} - {}'.format(trainX.max(), trainX.min(), testX.max(), testX.min()))

# Class labels for Fashion-MNIST

class\_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',

'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

# Display some loaded image data with labels

titles = [class\_names[label] for label in trainY[:9]]

display\_img(trainX[:9], titles)

# Expand dimensions for CNN input (28x28 grayscale to 28x28x1)

trainX = np.expand\_dims(trainX, axis=-1)

testX = np.expand\_dims(testX, axis=-1)

# Normalize the image data

trainX = trainX / 255.0

testX = testX / 255.0

# Investigate updated X

print('trainX.shape: {}, testX.shape: {}'.format(trainX.shape, testX.shape))

print('trainX.dtype: {}, testX.dtype: {}'.format(trainX.dtype, testX.dtype))

print('trainX.Range: {} - {}, testX.Range: {} - {}'.format(trainX.max(), trainX.min(), testX.max(), testX.min()))

# Turn Y into one-hot encoding correctly (num\_classes=10)

trainY = to\_categorical(trainY, num\_classes=10)

testY = to\_categorical(testY, num\_classes=10)

# Investigate updated Y

print('trainY.shape: {}, testY.shape: {}'.format(trainY.shape, testY.shape))

print('trainY.dtype: {}, testY.dtype: {}'.format(trainY.dtype, testY.dtype))

print(trainY[:5])

# Build the fully connected neural network model

inputs = Input((28, 28, 1), name='InputLayer')

x = Flatten()(inputs)

x = Dense(512, activation='relu')(x)

x = Dense(256, activation='relu')(x)

x = Dense(128, activation='relu')(x)

outputs = Dense(10, activation='softmax', name='OutputLayer')(x)

model = Model(inputs, outputs, name='Fashion-Multi-Class-Classifier')

model.summary()

# Compile the model

model.compile(loss='categorical\_crossentropy', metrics=['accuracy'])

# Train the model

model.fit(trainX, trainY, batch\_size=32, validation\_split=0.1, epochs=10)

# Evaluate model performance

model.evaluate(testX, testY)

# Predict Y values

predictY = model.predict(testX)

# Print original and predicted Y values

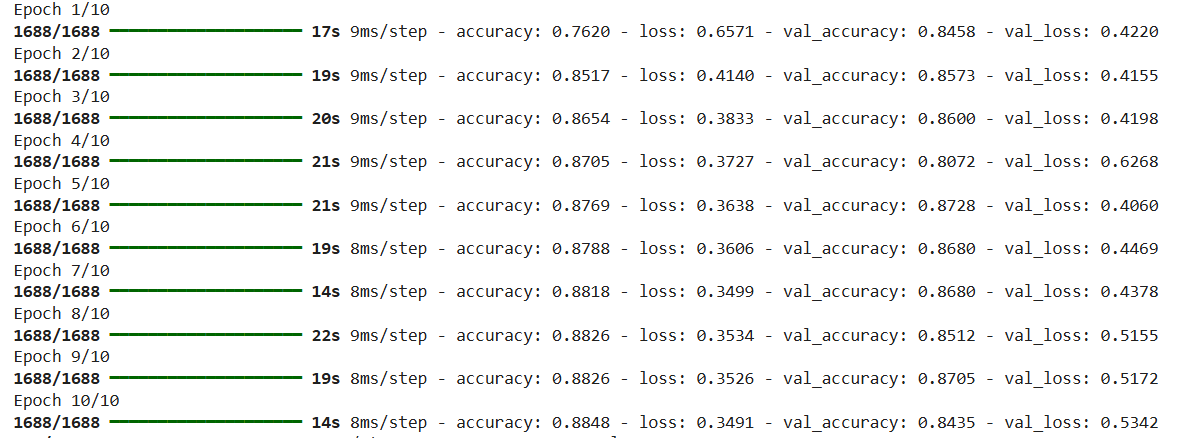
print('OriginalY PredictedY')

print('========= ===========')

for i in range(10):

print(np.argmax(testY[i]), '\t\t', np.argmax(predictY[i]))

Output:



Train and test CNN:

from tensorflow.keras.datasets import fashion\_mnist

import matplotlib.pyplot as plt

import numpy as np

from tensorflow.keras.utils import to\_categorical

from tensorflow.keras.layers import Input, Flatten, Conv2D

from tensorflow.keras.models import Model

# Helper function to display images

def display\_img(img\_set, title\_set):

n = len(title\_set)

for i in range(n):

plt.subplot(3, 3, i + 1)

plt.imshow(img\_set[i], cmap='gray')

plt.title(title\_set[i])

plt.show()

plt.close()

# Load the Fashion-MNIST dataset

(trainX, trainY), (testX, testY) = fashion\_mnist.load\_data()

# Investigate loaded data

print('trainX.shape: {}, trainY.shape: {}, testX.shape: {}, testY.shape: {}'.format(trainX.shape, trainY.shape, testX.shape, testY.shape))

print('trainX.dtype: {}, trainY.dtype: {}, testX.dtype: {}, testY.dtype: {}'.format(trainX.dtype, trainY.dtype, testX.dtype, testY.dtype))

print('trainX.Range: {} - {}, testX.Range: {} - {}'.format(trainX.max(), trainX.min(), testX.max(), testX.min()))

# Class labels for Fashion-MNIST

class\_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',

'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

# Display some loaded image data with labels

titles = [class\_names[label] for label in trainY[:9]]

display\_img(trainX[:9], titles)

# Expand dimensions for CNN input (28x28 grayscale to 28x28x1)

trainX = np.expand\_dims(trainX, axis=-1)

testX = np.expand\_dims(testX, axis=-1)

# Normalize the image data

trainX = trainX / 255.0

testX = testX / 255.0

# Investigate updated X

print('trainX.shape: {}, testX.shape: {}'.format(trainX.shape, testX.shape))

print('trainX.dtype: {}, testX.dtype: {}'.format(trainX.dtype, testX.dtype))

print('trainX.Range: {} - {}, testX.Range: {} - {}'.format(trainX.max(), trainX.min(), testX.max(), testX.min()))

# Turn Y into one-hot encoding correctly (num\_classes=10)

trainY = to\_categorical(trainY, num\_classes=10)

testY = to\_categorical(testY, num\_classes=10)

# Investigate updated Y

print('trainY.shape: {}, testY.shape: {}'.format(trainY.shape, testY.shape))

print('trainY.dtype: {}, testY.dtype: {}'.format(trainY.dtype, testY.dtype))

print(trainY[:5])

# Build the convolutional neural network model

num\_classes = 10

inputs = Input((28, 28, 1))

x = Conv2D(32, kernel\_size=(3, 3), padding = 'same', activation='relu')(inputs) # 32 filters

x = Conv2D(64, kernel\_size=(3, 3), padding = 'same', activation='relu')(x) # 64 filters

x = Conv2D(128, kernel\_size=(3, 3), padding = 'same', activation='relu')(x) # 128 filters

x = Flatten()(x)

outputs = Dense(num\_classes, activation='softmax', name="OutputLayer")(x)

model = Model(inputs, outputs, name="CNN\_Classifier")

model.summary()

# Compile the model

model.compile(loss='categorical\_crossentropy', metrics=['accuracy'])

# Train the model

model.fit(trainX, trainY, batch\_size=32, validation\_split=0.1, epochs=10)

# Evaluate model performance

model.evaluate(testX, testY)

# Predict Y values

predictY = model.predict(testX)

# Print original and predicted Y values

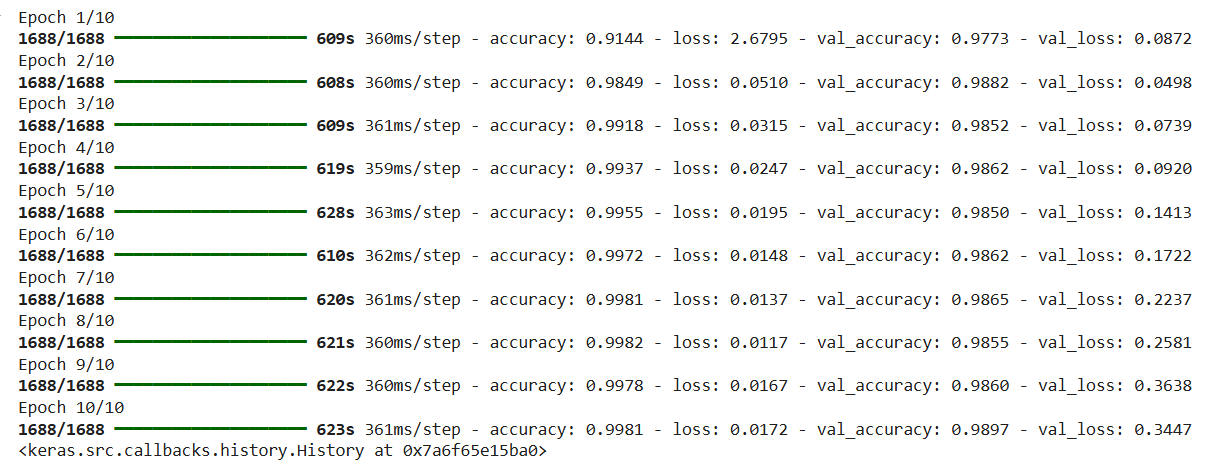
print('OriginalY PredictedY')

print('========= ===========')

for i in range(10):

print(np.argmax(testY[i]), '\t\t', np.argmax(predictY[i]))

Output:



Explanation:

From FCNN and CNN we can observe that the performance of CNN is better than FCNN as the accuracy for CNN is 99%(training level) and 98%(validation level) whereas for FCNN the accuracy is 88%%(training level) and 84%(validation level).

**Ques:3** Build a CNN having a pre-trained MobileNet as backbone to classify 10

classes.

**Ans:3**

from tensorflow.keras.applications import MobileNet

from tensorflow.keras.layers import Flatten, Dense

from tensorflow.keras.models import Model

mobilenet\_model = mobilenet.MobileNet()

mobilenet\_model.summary()

# Load mobilenet with pretrained weights

mobilenet\_model = mobilenet.MobileNet(input\_shape = (224, 224, 3), weights = 'imagenet', include\_top = False)

# Build a new model based on pre-trained mobilenet

inputs = mobilenet\_model.inputs

x = mobilenet\_model.output

x = Flatten()(x)

x = Dense(256, activation = 'relu')(x)

outputs = Dense(10, activation = 'softmax')(x)

model = Model(inputs, outputs, name = 'NewModel')

model.summary()

**Ques:4** Train and test your CNN having a pre-trained MobileNet as backbone to

classify images of the CIFAR-10 dataset. Discuss your results by comparing

performance between transfer\_learning + fine tuning and only transfer

learning.

**Ans:4**

To train and test a CNN with a pre-trained MobileNet model on the CIFAR-10 dataset, we need to make a few adjustments. CIFAR-10 consists of 32x32 RGB images, but MobileNet expects 224x224 RGB images. We will need to resize the images to match MobileNet's expected input size. Additionally, we will use the pre-trained MobileNet model as the feature extractor and build a new classification head to suit CIFAR-10's 10 classes.  
The code is:

import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.applications import MobileNet

from tensorflow.keras.datasets import cifar10

from tensorflow.keras.utils import to\_categorical

from tensorflow.keras.preprocessing.image import img\_to\_array, array\_to\_img

# Load CIFAR-10 dataset

(trainX, trainY), (testX, testY) = cifar10.load\_data()

# Normalize the image data to the range [0, 1]

trainX = trainX.astype('float32') / 255.0

testX = testX.astype('float32') / 255.0

# Resize the images to 224x224 (MobileNet input size)

trainX\_resized = tf.image.resize(trainX, (224, 224))

testX\_resized = tf.image.resize(testX, (224, 224))

# Convert labels to one-hot encoding

trainY = to\_categorical(trainY, num\_classes=10)

testY = to\_categorical(testY, num\_classes=10)

# Load the pre-trained MobileNet model without the top layer

mobilenet\_model = MobileNet(input\_shape=(224, 224, 3), weights='imagenet', include\_top=False)

# Freeze the layers of MobileNet

for layer in mobilenet\_model.layers:

layer.trainable = False

# Build a new model based on the pre-trained MobileNet

inputs = mobilenet\_model.inputs

x = mobilenet\_model.output

x = layers.GlobalAveragePooling2D()(x) # Global Average Pooling layer to reduce spatial dimensions

x = layers.Dense(256, activation='relu')(x) # Fully connected layer

outputs = layers.Dense(10, activation='softmax')(x) # 10 classes for CIFAR-10

model = models.Model(inputs, outputs)

# Compile the model

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# Train the model

model.fit(trainX\_resized, trainY, batch\_size=64, epochs=10, validation\_split=0.1)

# Evaluate the model on the test dataset

test\_loss, test\_acc = model.evaluate(testX\_resized, testY)

print(f"Test accuracy: {test\_acc:.4f}")

# Make predictions on the test set

predictions = model.predict(testX\_resized)

# Display the first 5 predictions and actual labels

for i in range(5):

print(f"Predicted: {predictions[i].argmax()}, Actual: {testY[i].argmax()}")