# Εφαρμογή της NAS σε πρόβλημα συσταδοποίησης (clustering)

(\* τα δεδομένα που χρησιμοποιούνται στο πείραμα είναι ειδικά επιλεγμένα για λόγους υπολογιστικής ευκολίας και κατανόησης του πειράματος) .

Παρακάτω θα παρουσιαστεί η διαδικασία αναζήτησης αρχιτεκτονικής για μοντέλο το οποίο πραγματοποιεί συσταδοποίηση προϊόντων μόδας σε 10 κατηγορίες. Για τους σκοπούς του προβλήματος (εκπαίδευση και αξιολόγηση) θα χρησιμοποιηθεί η βάση Fashion-MNIST η οποία περιέχει 70000 εικόνες (28x28 pixels, greyscaled) με αντίστοιχες ετικέτες από τις 10 κατηγορίες.

Παρακάτω φαίνεται ένα σενάριο εκπαίδευσης του μοντέλου (pipeline), το οποίο στη συνέχεια θα βελτιστοποιηθεί με τη βοήθεια της NAS :

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, διάγραμμα, σχεδίαση

Περιγραφή που δημιουργήθηκε αυτόματα

**1.**

**Παρατίθεται ο κώδικας (με ενδείξεις και σημειώσεις, που κατασκευάζει και εκπαιδεύει αυτό το μοντέλο**.

From tensorflow import keras

import numpy as np

from keras.layers import Conv2D,MaxPooling2D, Flatten, Dense

from keras.models import Sequential

from keras.datasets import fashion\_mnist

from keras.utils import to\_categorical

NUM\_CLASSES = 10

def load\_data():

# χωρίζουμε της εικόνες σε αυτές που θα χρησιμοποιηθούν για την εκπαίδευση και σε αυτές που θα χρησιμοποιηθούν για την αξιολόγηση του μοντέλου

(x\_train,y\_train),(x\_test,y\_test)=fashion\_mnist.load\_data()

# οι εικόνες είναι greyscaled συνεπώς τα pixels παίρνουν τιμές μεταξύ 0 και 255. Κανονικοποιούμε μεταξύ των τιμών 0 και 1 έτσι ώστε το μοντέλο μας να εκπαιδεύεται αποτελεσματικότερα

x\_train = x\_train.astype(“float32”)/255.0

x\_test = x\_test.astype(“float32”)/255.0

# για την πράξη της συνέλιξης (convolution) χρειάζεται και μια τρίτη διάσταση της εικόνας που σηματοδοτεί το είδος της (ασπρόμαυρες/με χρώμα)

x\_train = np.expand\_dims(x\_train,-1).astype(“float32”)

x\_test = np.expand\_dims(x\_test,-1).astype(“float32”)

# εκχωρούμε τις ετικέτες κάθε εικόνας στην αντίστοιχη από τις 10 κατηγορίες που ορίσαμε ως διάνυσμα 1x10, οπού το διάνυσμα θα περιέχει μηδενικά και 1 στην θέση (index) της αντίστοιχης κατηγορίας

y\_train = to\_categorical(y\_train, NUM\_CLASSES)

y\_test = to\_categorical(y\_test, NUM\_CLASSES)

return (x\_train,x\_test),(y\_train,y\_test)

def build\_model():

# ορίζουμε το σειριακό σενάριο εκπαίδευσης του μοντέλου (pipeline)

model = Sequential([

Conv2D(32,3,activation=’relu’),

Conv2D(64,3,activation=’relu’),

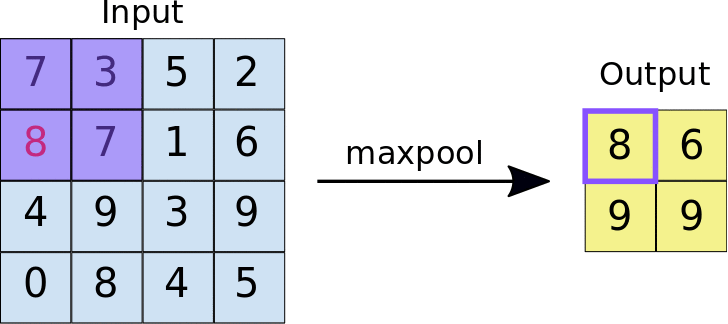
*Η λειτουργία της συνέλιξης (convolution) σε μια εικόνα (δηλαδή ενός πίνακα από pixels) γίνεται για την εξαγωγή χαρακτηριστικών όπως οι ακμές, οι γωνίες, τα όρια κ.α. με σκοπό αυτά τα χαρακτηριστικά με τη σειρά τους να βοηθήσουν στην εξαγωγή συμπερασμάτων. (Π.χ. αν οι ακμές σε δύο σημεία της εικόνας είναι κυρτές, τότε μάλλον πρόκειται η εικόνα να απεικονίζει μια μπλούζα). Η λειτουργία αυτή φαίνεται και στο παρακάτω animation :*

Εικόνα που περιέχει ορθογώνιο παραλληλόγραμμο, τετράγωνο, σχεδίαση, κύβος

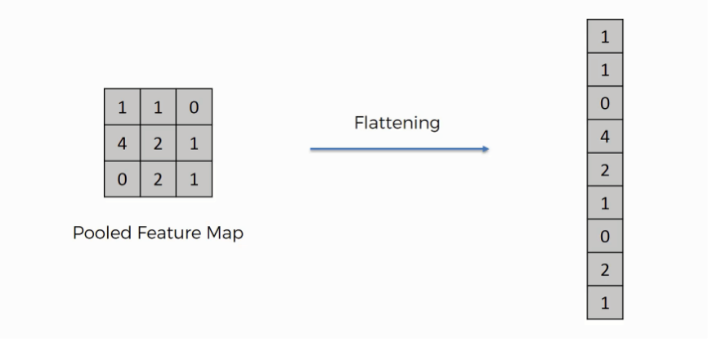
Περιγραφή που δημιουργήθηκε αυτόματα

MaxPooling2D(),

*Με τη σειρά της η λειτουργία MaxPooling κάνει ακόμη πιο εμφανή τα χαρακτηριστικά που εξίχθησαν από την προηγούμενη συνέλιξη :*

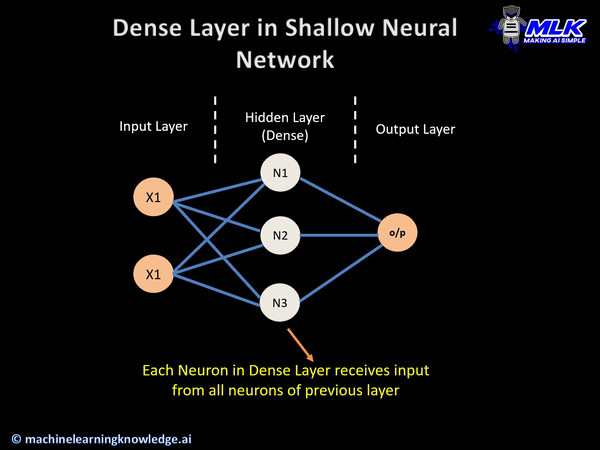


Flatten(),



Dense(100,activation = ‘relu’),

*Ένα dense (density) επίπεδο σε ένα δίκτυο, επιστρέφει τις πιθανότητες (χρησιμοποιώντας την συνάρτηση ενεργοποίησης) μια είσοδος να φέρει μια συγκεκριμένη ιδιότητα. Στην περίπτωσή μας είναι η ιδιότητα αυτή είναι η κατηγορία στην οποία ανήκει η είσοδος :*



Dense(10,activation = ‘softmax’)

])

# προσθέτουμε έναν βελτιστοποιητή για τα βάρη του δικτύου της αρχιτεκτονικής με βάση έναν ρυθμό βελτιστοποίησης σε κάθε βήμα

optimizer = keras.optimizers.Adam(learning\_rate=0.01)

# ορίζουμε την συνάρτηση κόστους με βάση την οποία θα εκτιμήσουμε την απόδοση του δικτύου της αρχιτεκτονικής

#Στη συγκεκριμένη περίπτωση επιλέγουμε την categorical crossentropy η οποια επιστρέφει ένα μέτρο (κόστος) της διαφοράς μεταξύ της εκτιμώμενης και της πραγματικής τιμής κάθε εισόδου (στην πρόβλεψη της κατηγορίας στην οποία ανήκει η είσοδος). Το κόστος αυτό είναι μικρό αν επιστραφεί τιμή κοντά στο 0 και μεγάλο αν επιστραφεί τιμή κοντά στο 1

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

return model

#ΕΚΠΑΙΔΕΥΣΗ ΤΟΥ ΜΟΝΤΕΛΟΥ

#1. Φόρτωση των δεδομένων με κλήση της αντίστοιχης συνάντησης

(x\_train,x\_test),(y\_train,y\_test) = load\_data()

#2. Κατασκευή του μοντέλου με κλήση της αντίστοιχης συνάρτησης

model = build\_model()

#3.Εκπαίδευση

#Σε αυτό το στάδιο πρέπει να ορίσουμε (τα δεδομένα εκπαίδευσης, πόσες εικόνες θα επεξεργάζεται το μοντέλο ταυτόχρονα, για πόσες επαναλήψεις θα εκπαιδεύσω το μοντέλο καθώς και τα δεδομένα εκτίμησης απόδοσης του μοντέλου) όπως φαίνεται και στα αντίστοιχα ορίσματα

model.fit(x\_train,y\_train,batch\_size=200,epochs=10, validation\_data=(x\_test,y\_test))

# η απώλεια (ή loss) είναι μια μετρική που αξιολογεί το πόσο καλά το μοντέλο προβλέπει τα δεδομένα δοκιμής σε σχέση με τις πραγματικές τους τιμές.

# η ακρίβεια είναι το ποσοστό των προβλέψεων που είναι σωστές σε σχέση με τις πραγματικές τιμές.

loss, acc = model.evaluate(x\_test,y\_test)

print(acc)

**2.**

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό, λογισμικό πολυμέσων

Περιγραφή που δημιουργήθηκε αυτόματα

Με το παραπάνω σενάριο βλέπουμε ότι το μοντέλο μας αξιολογείται με ακρίβεια 0.8980000019073486, κοντά δηλαδή στο 90%. Αυτό σημαίνει ότι για μια συγκεκριμένη είσοδο (εικόνα) το μοντέλο θα την κατατάξει στην σωστή κατηγορία με πιθανότητα περίπου 0.9.

Πώς όμως μπορούμε να είμαστε σίγουροι ότι η αρχιτεκτονική που χρησιμοποιήθηκε είναι η καταλληλότερη για την συγκεκριμένη διεργασία και πώς μπορούμε να μελετήσουμε την παραπάνω απορία με έναν γρήγορο και αποτελεσματικό τρόπο;

Την απάντηση έρχεται να δώσει η NAS όπως θα φανεί παρακάτω :

**3.1 Ορισμός χώρου αναζήτησης**

Όπως αναφέρθηκε και νωρίτερα, για να αναζητήσουμε για καλύτερη αρχιτεκτονική, θα πρέπει να διαμορφώσουμε ένα χώρο ο οποίος θα περιέχει όλες τις πιθανές αρχιτεκτονικές. Κάτι τέτοιο μπορεί να συμβεί όπως φαίνεται παρακάτω, όπου για τις διεργασιές (της συνέλιξης, της διαμόρφωσης πιθανότητας του ανήκειν και του ρυθμού με βάση τον οποίο το μοντέλο εκπαιδεύεται) στο σενάριο εκπαίδευσης που παραθέσαμε νωρίτερα, δίνονται εναλλακτικές. Με αυτό τον τρόπο έχουμε στην διάθεσή μας 3\*3\*2\*2\*3 = 108 αρχιτεκτονικές στον global χώρο αναζήτησης, καθεμία από τις οποίες δοκιμάζεται με διαφορετικούς ρυθμούς εκμάθησης (οι ρυθμοί εκμάθησης επιλέγονται τυχαία στο συνεχές διάστημα (0.001-0.01)) :

search\_space = {

"filter\_size\_c1" : {'\_type' : 'choice', '\_value': [32,64,128]},

"filter\_size\_c2" : {'\_type' : 'choice', '\_value': [32,64,128]},

Εικόνα που περιέχει διάγραμμα, κείμενο, Σχέδιο, τεχνικό σχέδιο

Περιγραφή που δημιουργήθηκε αυτόματα

"kernel\_size\_c1" : {'\_type' : 'choice', '\_value': [3,5]},

"kernel\_size\_c2" : {'\_type' : 'choice', '\_value': [3,5]},

# Ο αριθμός των μονάδων (nb units) σχετίζεται με την διεργασία Dense που περιγράφηκε παραπάνω και είναι ο αριθμός των κόμβων που θα έχει η διεργασία στο κρυφό της επίπεδο.

"nb\_units" : {'\_type' : 'choice', '\_value': [80, 100, 120]},

"learning\_rate" : {'\_type' : 'uniform', '\_value': [0.001,0.01]}

}

**3.2**

Τώρα που ορίστηκε ο χώρος, αναδιαμορφώνουμε τον κώδικά μας έτσι ώστε πλέον να εκπαιδεύει το μοντέλο με βάση τις λειτουργίες που του παρέχονται κατά τη διάρκεια του πειράματος.

def build\_model(params):  
 # ορίζουμε το σειριακό σενάριο εκπαίδευσης του μοντέλου (pipeline)  
 model = Sequential([  
 Conv2D(**params['filter\_size\_c1'], params['kernel\_size\_c1']**, activation='relu'),  
 Conv2D(**params['filter\_size\_c2'], params['kernel\_size\_c2']**, activation='relu'),  
 MaxPooling2D(),  
 Flatten(),  
 Dense(**params['nb\_units']**, activation='relu'),  
 Dense(10, activation='softmax')  
 ])  
  
 # προσθέτουμε έναν βελτιστοποιητή για τα βάρη του δικτύου μας με βάση έναν ρυθμό βελτιστοποίησης σε κάθε βήμα  
 optimizer = keras.optimizers.Adam**(learning\_rate=params['learning\_rate'])**  
  
 # ορίζουμε την συνάρτηση κόστους με βάση την οποία θα εκτιμήσουμε την απόδοση του δικτύου μας  
 #Στη συγκεκριμένη περίπτωση επιλέγουμε την categorical crossentropy η οποια μας επιστρέφει ένα μέτρο (κόστος) της διαφοράς μεταξύ της εκτιμώμενης και της πραγματικής τιμής κάθε εισόδου (στην πρόβλεψη της κατηγορίας στην οποία ανήκει η είσοδος). Το κόστος αυτό είναι μικρό αν επιστραφεί τιμή κοντά στο 0 και μεγάλο αν επιστραφεί τιμή κοντά στο 1  
 model.compile(optimizer=optimizer, loss='categorical\_crossentropy', metrics=['accuracy'])  
  
 return model  
# κλάση υπεύθυνη να αναφέρει κατά το πείραμα την ακρίβεια του μοντέλου μετά το τέλος κάθε εποχής (epoch) εκπαίδευσής του  
class SendMetrics(Callback):  
 def on\_epoch\_end(self, epoch, logs=None):  
 nni.report\_intermediate\_result(logs['val\_accuracy'])  
  
#ΕΚΠΑΙΔΕΥΣΗ ΤΟΥ ΜΟΝΤΕΛΟΥ  
def run(params):  
 #1. Φόρτωση των δεδομένων με χρήση της αντίστοιχης συνάντησης  
 (x\_train,x\_test),(y\_train,y\_test) = load\_data()  
 #2. Κατασκευή του μοντέλου με κλήση της αντίστοιχης συνάρτησης  
 model = build\_model(params)  
 #3.Εκπαίδευση  
 #Σε αυτό το στάδιο πρέπει να ορίσουμε (τα δεδομένα εκπαίδευσης, πόσες εικόνες θα επεξεργάζεται το μοντέλο ταυτόχρονα, για πόσες επαναλήψεις θα εκπαιδεύσω το μοντέλο, τα δεδομένα εκτίμησης απόδοσης του μοντέλου)

model.fit(x\_train,y\_train,batch\_size=200,epochs=10,  
 validation\_data=(x\_test,y\_test),callbacks=[SendMetrics()])

# η απώλεια (ή loss) είναι μια μετρική που αξιολογεί το πόσο καλά το μοντέλο προβλέπει τα δεδομένα δοκιμής σε σχέση με τις πραγματικές τους τιμές.

# η ακρίβεια είναι το ποσοστό των προβλέψεων που είναι σωστές σε σχέση με τις πραγματικές τιμές.  
 loss, acc = model.evaluate(x\_test,y\_test)

# αφού ολοκληρωθεί η εκπαίδευση, η τελική ακρίβεια αξιολογείται και αναφέρεται ως τελικό αποτέλεσμα  
 nni.report\_final\_result(acc)  
 print(acc)  
if \_\_name\_\_ == '\_\_main\_\_':  
 try:

# μετά την εκπαίδευση και την εκτίμηση της εκάστοτε αρχιτεκτονικής, με την στρατηγική επόμενης επιλογής, επιλέγουμε την επόμενη αρχιτεκτονική από τον χώρο αναζήτησης  
 params = nni.get\_next\_parameters()  
 run(params) except Exception:  
 raise

**3.3**

Έχοντας ορίσει τον χώρο αναζήτησης και τη στρατηγική, είμαστε πλέον σε θέση να διαμορφώσουμε το πείραμα της NAS, με τη βοήθεια του εργαλείου Neural Network Intelligence (NNI) :

from nni.experiment import Experiment

#διαδικαστικές δηλώσεις για την διεξαγωγή του πειράματος

experiment = Experiment('local')

experiment.config.trial\_command = 'python fashionmnistmodel.py'

experiment.config.trial\_code\_directory = '.'

experiment.config.search\_space = search\_space

experiment.config.tuner.name = 'TPE'

experiment.config.trial\_concurrency = 1

#αριθμός των αρχιτεκτονικών που θα δοκιμαστούν (μικρός για υπολογιστική ευκολία)

experiment.config.max\_trial\_number = 5

**4.**

Τέλος, εκτελούμε το πείραμα :

experiment.run(web server listening portal)

Παρακάτω βλέπουμε τα αποτελέσματα στις ακρίβειες από τις πέντε πιο επιτυχείς αρχιτεκτονικές:

**Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό

Περιγραφή που δημιουργήθηκε αυτόματα**

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά, σχεδίαση

Περιγραφή που δημιουργήθηκε αυτόματαΒλέπουμε ότι η αρχιτεκτονική με τα παραπάνω χαρακτηριστικά, δίνει την καλύτερη ακρίβεια 0,914 η οποία είναι και καλύτερη από αυτήν που παρουσιάστηκε νωρίτερα ( 0,898 ).

# Βιβλιογραφία

<https://arxiv.org/abs/2301.08727v2>

<https://www.youtube.com/watch?v=td820ts6gUU&t=3687s>