Νευρωνικά Δίκτυα - Βαθιά Μάθηση
3ο Παραδοτέο Εργασίας

Τουλκερίδης Νικόλαος
ΗΜΜΥ 10718

ΕΙΣΑΓΩΓΗ

Η παρούσα εργασία επικεντρώνεται στη σχεδίαση και αξιολόγηση ενός RBF νευρωνικού δικτύου, το οποίο θα εκπαιδευτεί ώστε να ταξινομεί είτε δύο επιλεγμένες κατηγορίες είτε και τις δέκα κατηγορίες της βάσης δεδομένων CIFAR-10. Η CIFAR-10 αποτελεί μια ευρέως χρησιμοποιούμενη βάση δεδομένων που περιλαμβάνει έγχρωμες, μικρού μεγέθους εικόνες ταξινομημένες σε 10 διαφορετικές κατηγορίες.

Για την ορθή κατασκευή του δικτύου, προηγείται προεπεξεργασία των δεδομένων, όπου εφαρμόζεται η μέθοδος Ανάλυσης Κύριων Συνιστωσών (PCA) για μείωση των διαστάσεων. Στη συνέχεια, εξετάζονται διάφορες παραμετροποιήσεις, όπως ο αριθμός των κέντρων του δικτύου, η μέθοδος επιλογής αυτών των κέντρων και ο τρόπος υπολογισμού της ακτίνας τους.

ΠΕΡΙΓΡΑΦΗ ΑΛΓΟΡΙΘΜΟΥ

Η εργασία αυτή επικεντρώνεται στη μελέτη των RBF νευρωνικών δικτύων στο σύνολο δεδομένων CIFAR-10, το οποίο περιλαμβάνει έγχρωμενες εικόνες ταξινομημένες σε 10 κατηγορίες. Τα RBF δίκτυα βασίζονται στις Radial Basis Functions (RBFs) ως συναρτήσεις ενεργοποίησης. Αυτές οι συναρτήσεις εξαρτώνται από την απόσταση των εισόδων από συγκεκριμένα σημεία, γνωστά ως “centers”. Τα “centers” αναπαριστούν τοπικά μοτίβα ή συστάδες στα δεδομένα και μπορούν να καθοριστούν είτε μέσω αλγορίθμων ομαδοποίησης, όπως το K-means, είτε κατά την εκπαίδευση του δικτύου. Όσο πιο κοντά βρίσκεται μια είσοδος σε κάποιο center, τόσο ισχυρότερα ενεργοποιείται η αντίστοιχη RBF συνάρτηση, υποδεικνύοντας ότι η είσοδος πιθανόν ανήκει σε μια ομάδα δεδομένων με παρόμοια χαρακτηριστικά.

Ένα τυπικό RBF νευρωνικό δίκτυο μπορεί να περιγραφεί ως εξής:

$$yk(x) = \sum_{i=1}^{M} w_{ki}\varphi_i(x) + b_k$$

όπου:

- yk(x) είναι η έξοδος του k-οστού νευρώνα στο output layer,
- w_{ki} είναι το βάρος που συνδέει το i-οστό RBF με το k-οστό output,
- $\varphi_i(x)$ είναι η έξοδος της i-οστής συνάρτησης RBF,
- b_k είναι το bias του k-οστού output.

Τα RBF νευρωνικά δίκτυα αποτελούνται από τρία layers:

- Το input layer, το οποίο δέχεται τα δεδομένα εισόδου.
- Το hidden layer (RBF layer), το οποίο περιλαμβάνει τις RBF συναρτήσεις ενεργοποίησης και μετρά την απόσταση των εισόδων από τα centers, πραγματοποιώντας μια μη-γραμμική χαρτογράφηση του χώρου.
- Το output layer, το οποίο συνδυάζει τις εξόδους του hidden layer με βάρη για να παράγει το τελικό αποτέλεσμα.

Η Gaussian συνάρτηση, που χρησιμοποιείται συχνά ως RBF, επιτρέπει την εισαγωγή μη-γραμμικών μετασχηματισμών. Αυτό είναι ιδιαίτερα χρήσιμο για την αναπαράσταση και διαχωρισμό σύνθετων, μη-γραμμικών περιοχών απόφασης. Η παράμετρος σ (διασπορά) ελέγχει το εύρος της ενεργής περιοχής της Gaussian. Μικρές τιμές σ κάνουν τη συνάρτηση να επικεντρώνεται σε στενές περιοχές γύρω από τα centers, ενώ μεγαλύτερες τιμές σ διευρύνουν την περιοχή επιρροής.

ΕΠΕΞΗΓΗΣΗ ΚΩΔΙΚΑ

```python
import numpy as np
import tensorflow as tf

from sklearn.decomposition import PCA
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.cluster import KMeans

from keras.models import Sequential
from keras.layers import Dense
from keras.initializers import RandomNormal
from keras.utils import to_categorical
from keras.datasets import cifar10
from keras.callbacks import EarlyStopping, LearningRateScheduler

import time
import seaborn as sns
import matplotlib.pyplot as plt
from itertools import product
```

Οι απαραίτητες βιβλιοθήκες για την υλοποίηση του κώδικα

```python
# Φόρτωση CIFAR-10
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train, x_test = x_train.astype("float32") / 255.0, x_test.astype("float32") / 255.0
y_train, y_test = to_categorical(y_train), to_categorical(y_test)
print("Shape of y_train:", y_train.shape)
print("Shape of y_test:", y_test.shape)

# Διαμόρφωση των δεδομένων (reshape για PCA)
x_train_flat = x_train.reshape(x_train.shape[0], -1)
x_test_flat = x_test.reshape(x_test.shape[0], -1)

# PCA: Διατήρηση πάνω από 90% της πληροφορίας
pca = PCA(0.95)
x_train_pca = pca.fit_transform(x_train_flat)
x_test_pca = pca.transform(x_test_flat)
```

Το φόρτωμα της cifar-10
Το one-hot encoding των labels ελέγχου. (μετατροπή της δεκαδικής μορφής ενός label σε 10χ1 διάνυσμα με 1 label-στής γραμμές και 0 όλες τις άλλες)
Το flatten των δεδομένων εισόδου
Η εφαρμογή PCA για διατήρηση του 95% της σημαντικής πληροφορίας του κάθε δείγματος

Η υλοποίηση της RBFNN κλάσης. Οι αρχικοποίησεις των κέντρων και των βήτα, που μελετήθηκαν. Η προσθήκη του denser layer 256 νευρώνων, ο Adam optimizer.
Το 5-fold cross validation, τα 1000 κέντρα και η τελική προπόνηση του μοντέλου με 150 εποχές.

```python
class RBFNN(tf.keras.Model):
    def __init__(self, n_centers, n_classes, input_dim, training_data):
        super(RBFNN, self).__init__()
        self.n_centers = n_centers

        # Υπολογισμός των RBF κέντρων με K-Means
        self.centers = self.init_centers_with_kmeans(training_data)
        # Initialize betas based on distances between centers
        self.betas = self.init_betas(self.centers)

        # Dense layer for classification
        self.dense1 = tf.keras.layers.Dense(
            256, activation="relu", kernel_initializer=tf.keras.initializers.HeNormal()
        )
        self.output_layer = tf.keras.layers.Dense(
            n_classes, activation="softmax", kernel_initializer=tf.keras.initializers.HeNormal()
        )
```

```python
    def init_centers_with_kmeans(self, data):
        """Υπολογισμός κέντρων με K-Means clustering."""
        kmeans = KMeans(n_clusters=self.n_centers, random_state=42)
        kmeans.fit(data)
        return tf.Variable(kmeans.cluster_centers_, trainable=True, dtype=tf.float32, name="centers")

    def init_betas(self, centers):
        # Compute distances between all centers
        distances = tf.reduce_mean(
            tf.norm(
                tf.expand_dims(centers, axis=0) - tf.expand_dims(centers, axis=1), axis=2
            ),
            axis=1
        )
        # Return betas based on distances
        return tf.Variable(1 / (2 * distances ** 2), trainable=True, name="betas")

    def rbf_layer(self, inputs):
        # Υπολογισμός των RBF συναρτήσεων
        dists = tf.reduce_sum((tf.expand_dims(inputs, 1) - self.centers) ** 2, axis=2)
        rbf_outputs = tf.exp(-self.betas * dists)
        return rbf_outputs

    def call(self, inputs):
        rbf_outputs = self.rbf_layer(inputs)
        x = self.dense1(rbf_outputs)
        outputs = self.output_layer(x)
        return outputs
```

```python
# Εκπαίδευση και αξιολόγηση με cross-validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)
train_accuracies, test_accuracies, train_times = [], [], []

for train_index, val_index in kf.split(x_train_pca):
    x_train_cv, x_val_cv = x_train_pca[train_index], x_train_pca[val_index]
    y_train_cv, y_val_cv = y_train[train_index], y_train[val_index]

    model_B = RBFNN(
        n_centers=1000,
        n_classes=10,
        input_dim=x_train_pca.shape[1],
        training_data=x_train_pca
    )

    optimizer = tf.keras.optimizers.Adam(learning_rate=1e-4)
    #optimizer = tf.keras.optimizers.AdamW(learning_rate=1e-4, weight_decay=1e-5)
    #optimizer = tf.keras.optimizers.RMSprop(learning_rate=1e-4)
    #optimizer = tf.keras.optimizers.SGD(learning_rate=1e-4, momentum=0.9)

    model_B.compile(optimizer=optimizer, loss="categorical_crossentropy", metrics=["accuracy"])
```

```python
    model_B.compile(optimizer=optimizer, loss="categorical_crossentropy", metrics=["accuracy"])

    # Εκκίνηση του χρόνου εκπαίδευσης
    start_time = time.time()

    # Εκπαίδευση
    history = model_B.fit(
        x_train_cv, y_train_cv,
        validation_data=(x_val_cv, y_val_cv),
        epochs=150,
        batch_size=64,
        verbose=1
    )
    # Αποθήκευση του μοντέλου
    model_B.save("rbfnn_model_FINAL.keras")

    # Καταγραφή χρόνου
    end_time = time.time()
    train_times.append(end_time - start_time)

    # Ακρίβεια στο σύνολο ελέγχου
    train_acc = max(history.history['accuracy'])
    val_acc   = max(history.history['val_accuracy'])

    train_accuracies.append(train_acc)
    test_accuracies .append(val_acc)
```

Η εκτύπωση των αποτελεσμάτων και των σωστών-λανθασμένων προβλέψεων.

```python
# Εμφάνιση των αποτελεσμάτων
plt.figure(figsize=(12, 6))
for i, idx in enumerate(samples_to_plot):
    plt.subplot(2, 3, i + 1)
    plt.imshow(x_test[idx])
    true_label = class_names[true_labels[idx]]
    pred_label = class_names[predicted_labels[idx]]
    if i < 3:
        plt.title(f"Correct\nTrue: {true_label}\nPredicted: {pred_label}", color='green')
    else:
        plt.title(f"Incorrect\nTrue: {true_label}\nPredicted: {pred_label}", color='red')
    plt.axis('off')


plt.tight_layout()
plt.show()
```

ΔΟΚΙΜΕΣ

Αρχικά, ξεκίνησα γράφοντας το μοντέλο στην πιο απλή μορφή του.

- Τα κέντρα του αρχικοποιούνται με τυχαίο τρόπο και είναι trainable ώστε το back propagation να μπορεί να τα μεταβάλλει.
- Τα βήτα αρχικοποιούνται με την τιμή 1 και είναι επίσης trainable.
- Το dense layer με 10 νευρώνες, όσες και οι κλάσεις μας.
- 100 κέντρα
- 50 εποχές σε 5-fold cross validation
- Adam optimizer

Μετά τη δοκιμή αυτή παρατηρήθηκε ότι το μοντέλο 'κολλάει' στο 9.5-10% για όλο το training.

```
Cross-validation Training Accuracies:  [0.1025250032544136, 0.10167499631643295, 0.1008249968290329, 0.10114999860525131, 0.10217
499732971191]
Cross-validation Validation Accuracies:  [0.1014999970793724, 0.10180000215768814, 0.1006999984383583, 0.10010000318288803, 0.101
09999775886536]
Average Training Time per Fold: 98.73 seconds
Test Accuracy: 10.00%
```

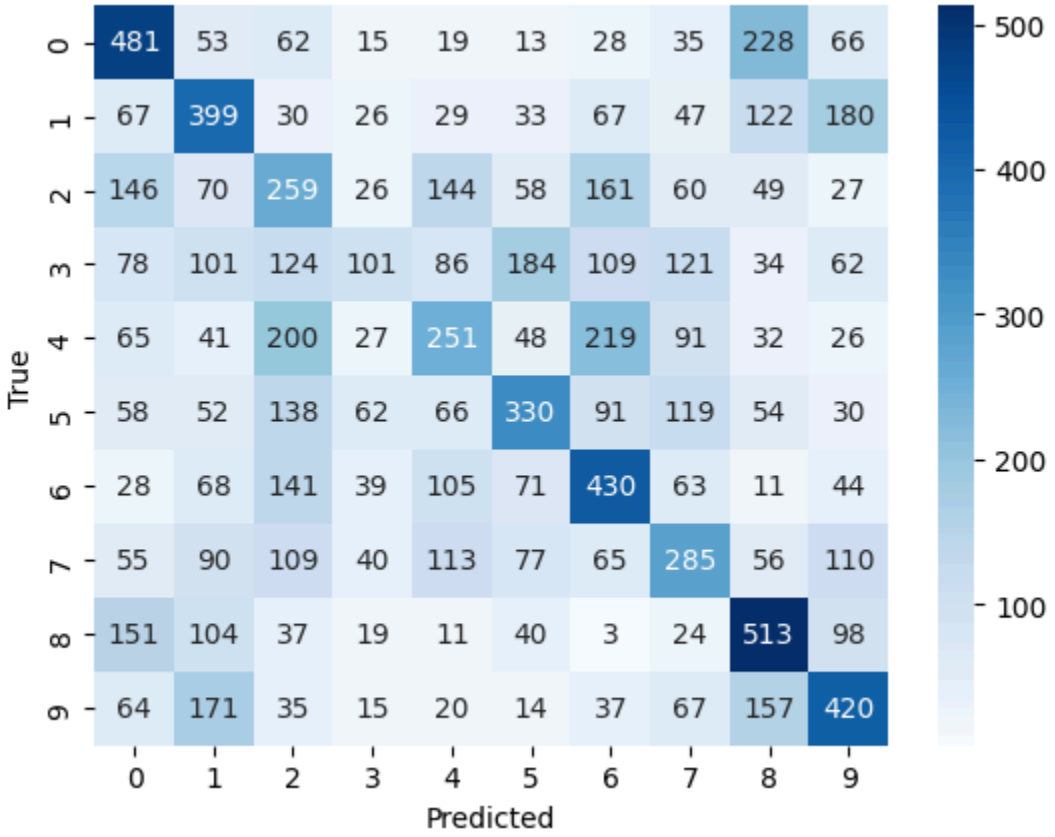
Δοκιμάστηκε η αύξηση των κέντρων σε 1000 και παρατηρήθηκαν τα ίδια αποτελέσματα.

Σε αυτό το σημείο, κρατώντας παράλληλα το πλήθος των κέντρων ίσο με 1000, άλλαξα τον τρόπο που αρχικοποιούνται τα *βήτα* θέτοντάς τα ίσα με τον αντίστροφο της διπλάσιας τετραγωνικής τιμής των αποστάσεων. Δηλαδή, αν ένα κέντρο βρίσκεται κοντά σε άλλα κέντρα, τότε η *βήτα* του είναι μεγαλύτερη, άρα η RBF γίνεται πιο στενή.
Εδώ επιτεύχθηκε **28,69% σε 16 λεπτά ανά fold**.

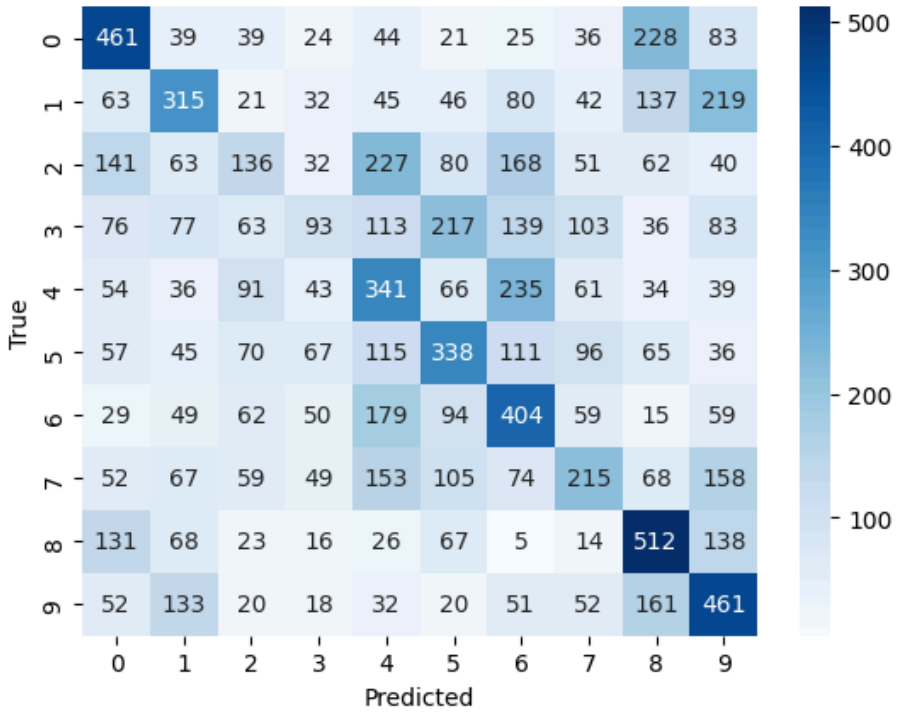
Για περαιτέρω βελτίωση της απόδοσης, δοκιμάστηκε ένας διαφορετικός τρόπος αρχικοποίησης των *κέντρων*, η τεχνική των k-μέσων.

Ο αλγόριθμος k-μέσων χωρίζει τα δεδομένα σε k ομάδες με βάση την ομοιότητα, αναθέτοντας κάθε σημείο στο πλησιέστερο από k αρχικά κέντρα και ενημερώνοντας τα κέντρα ως τους μέσους όρους των ομάδων. Η διαδικασία επαναλαμβάνεται έως ότου σταθεροποιηθούν τα κέντρα, με στόχο την ελαχιστοποίηση της συνολικής ενδο-ομαδικής απόστασης.

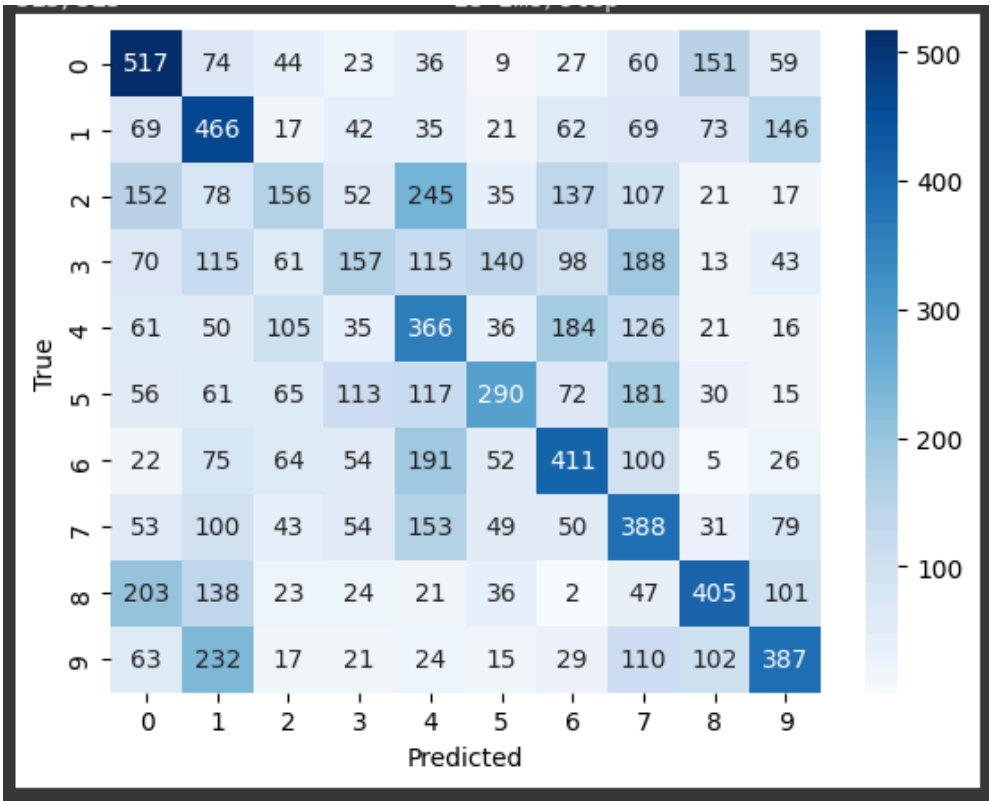
34.69% σε 25 λεπτά ανά fold



Έπειτα, δοκιμάστηκε η προεπιλεγμένη μεταβολή του learning rate. Ανά διαστήματα 20 εποχών, για συνολικά 80 εποχές, γινόντουσαν οι εξής μεταβολές:
0<εποχές<20: 10^(-4)
20<εποχές<40: 0.5*10^(-4)
40<εποχές<60: 10^(-5)
60<εποχές<80: 10^(-6)
Μετρήθηκε 32.76% με 1664” ανά fold. Καμία σημαντική βελτίωση, οπότε δεν προστέθηκε στο τελικό μοντέλο.



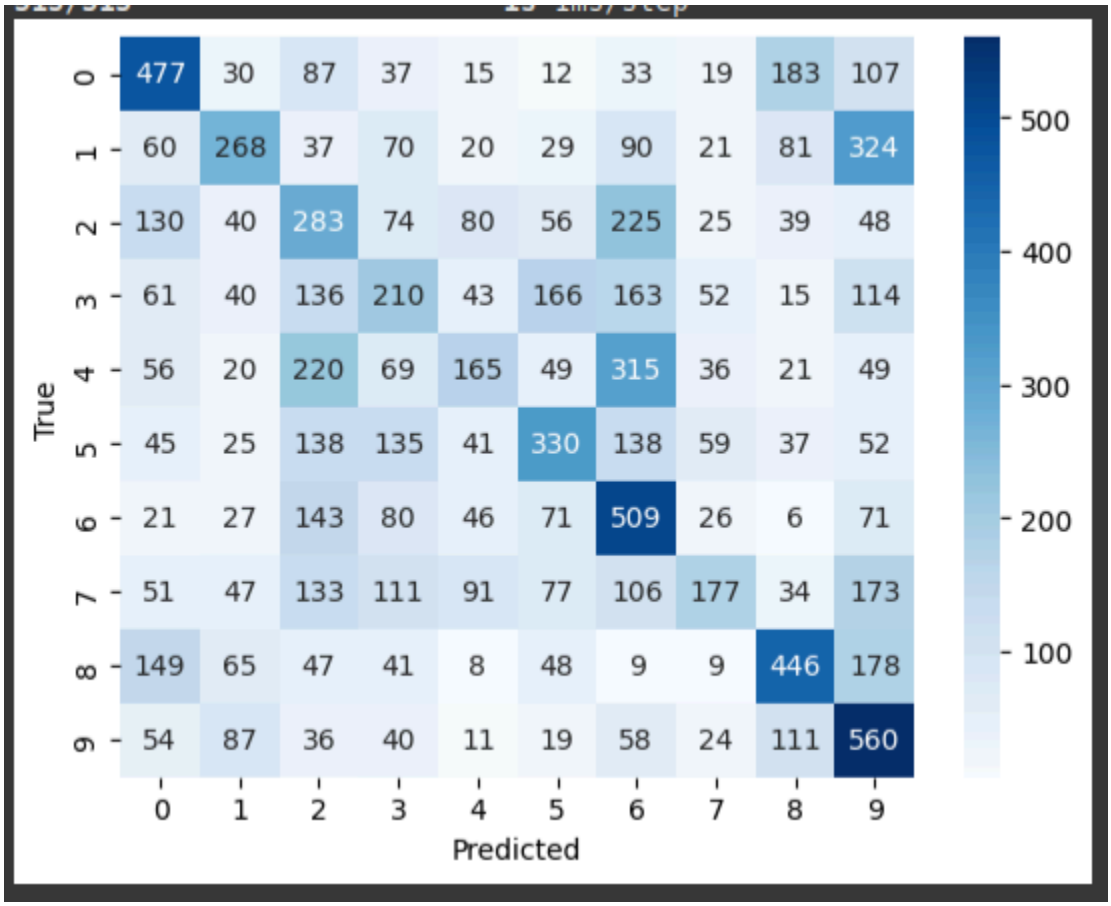
Αξίζει να σημειωθεί ότι δοκιμάστηκε το μοντέλο να έχει 2000 κέντρα αλλά δεν παρατηρήθηκε σημαντική βελτίωση της απόδοσης.
35.43% με 30' ανά fold



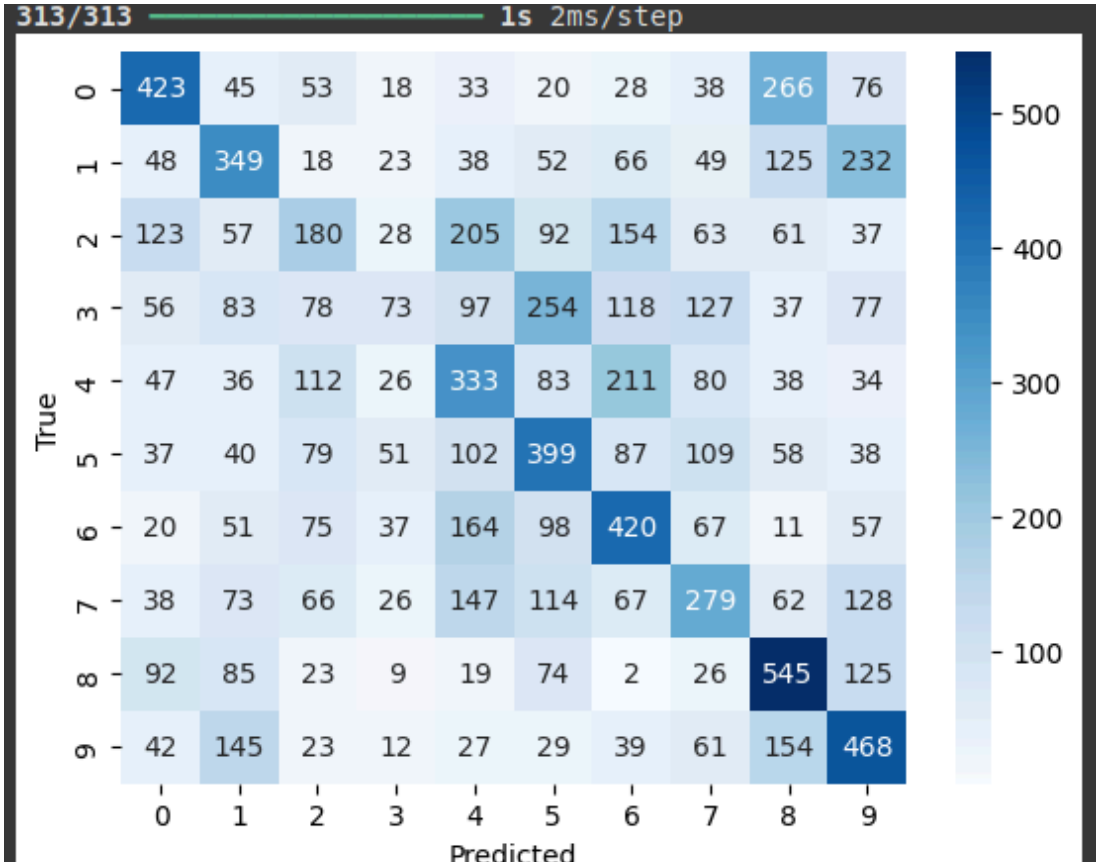
Διατηρώντας πλέον σταθερά τα 1000 κέντρα, 0.95 pca, 80 εποχές με 5-fold cross validation, k-means, betas=1/(2*dist^2), δοκιμάστηκαν διαφορετικοί optimizers για το μοντέλο:

SGD με momentum: 31.07% με 14 λεπτά ανά fold

RMS prop: 34.25% με 13.5 λεπτά ανά fold

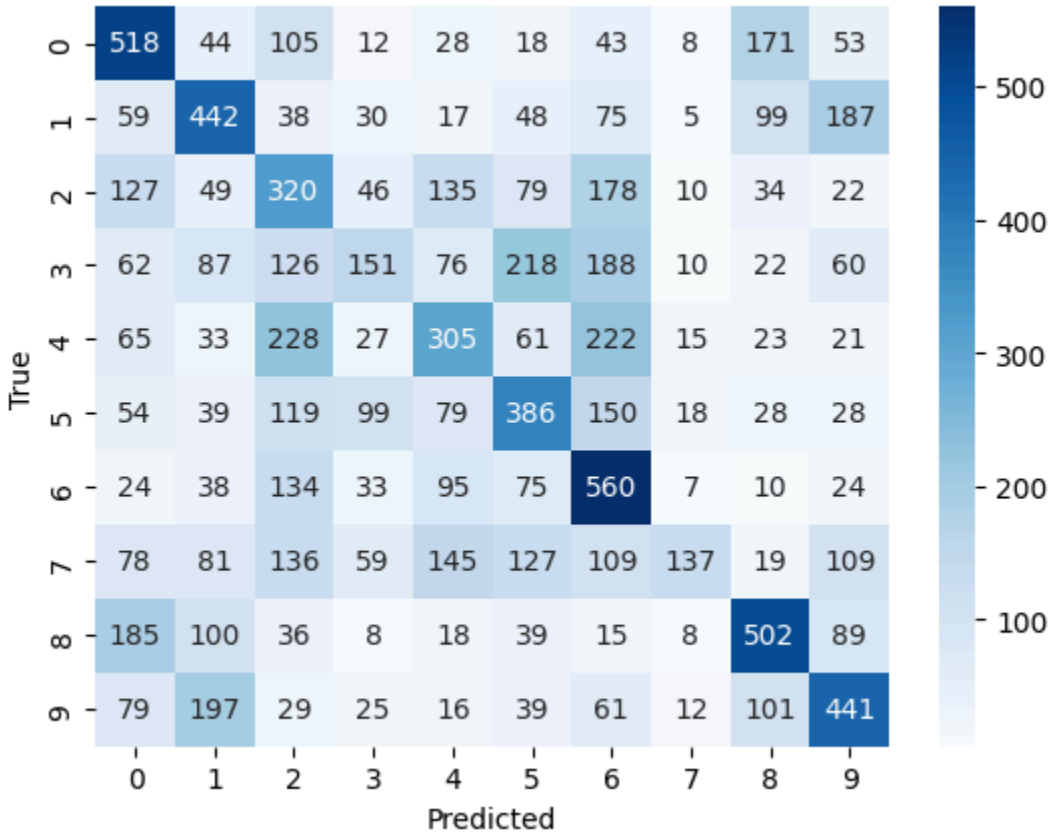


AdamW (Weight Decay): 34.69% με 15.1 λεπτά ανά fold

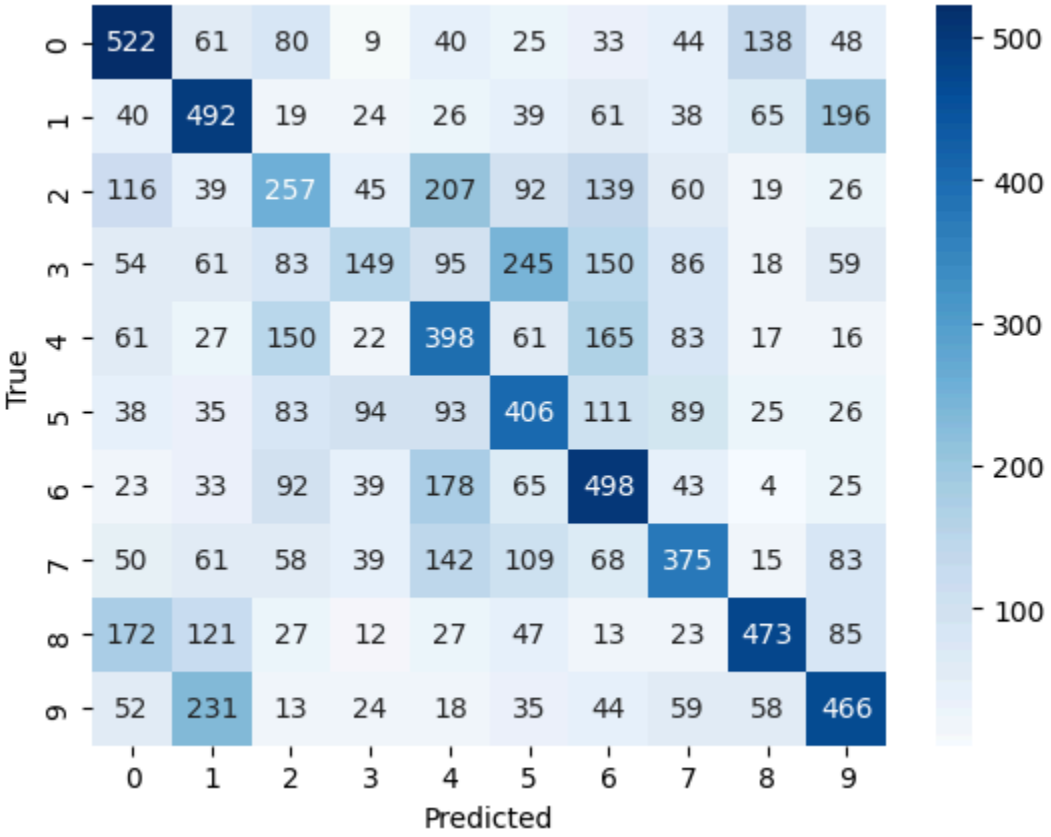


Δεν παρατηρήθηκε πάλι κάποια σημαντική αύξηση. Επιλέγεται ο αρχικός optimizer, Adam με σταθερό learning rate 10^(-4).

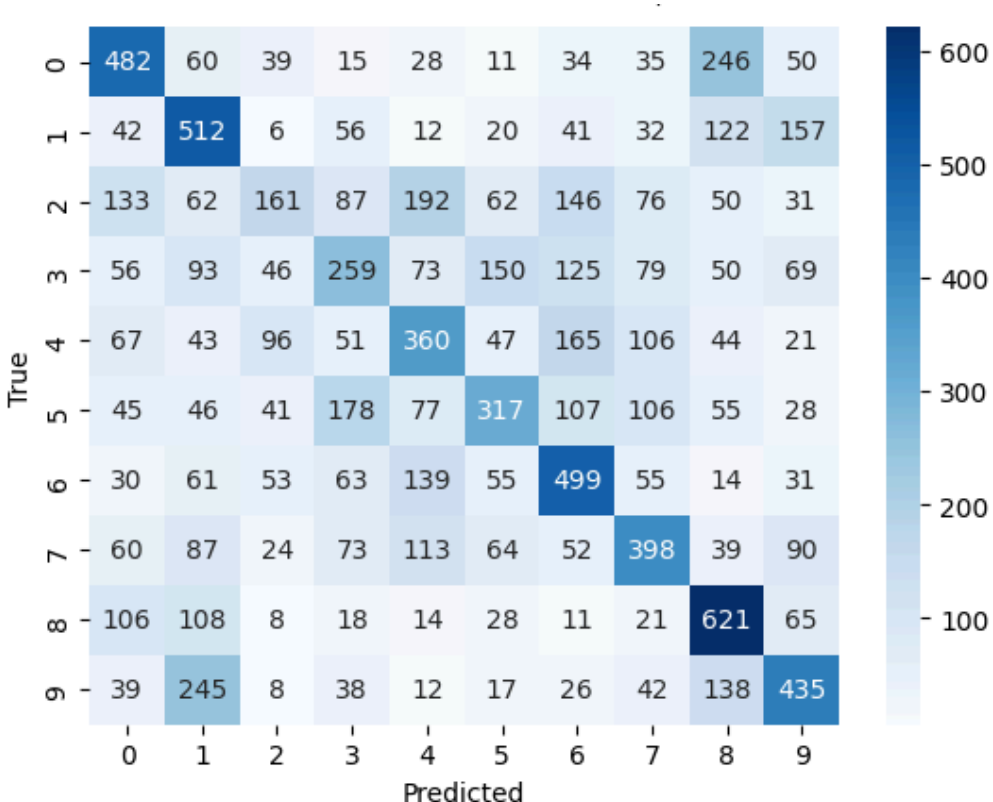
Επιπλέον, δοκιμάστηκε η προσθήκη ενός dense layer ανάμεσα στο rbf layer και στο output layer των 10 νευρώνων.
Αρχικά με 128 νευρώνες, και HeNormal() αρχικοποίηση.
37,62% με 35 λεπτά ανά fold



Με 256 νευρώνες: 40.36% με 28 λεπτά ανά fold.



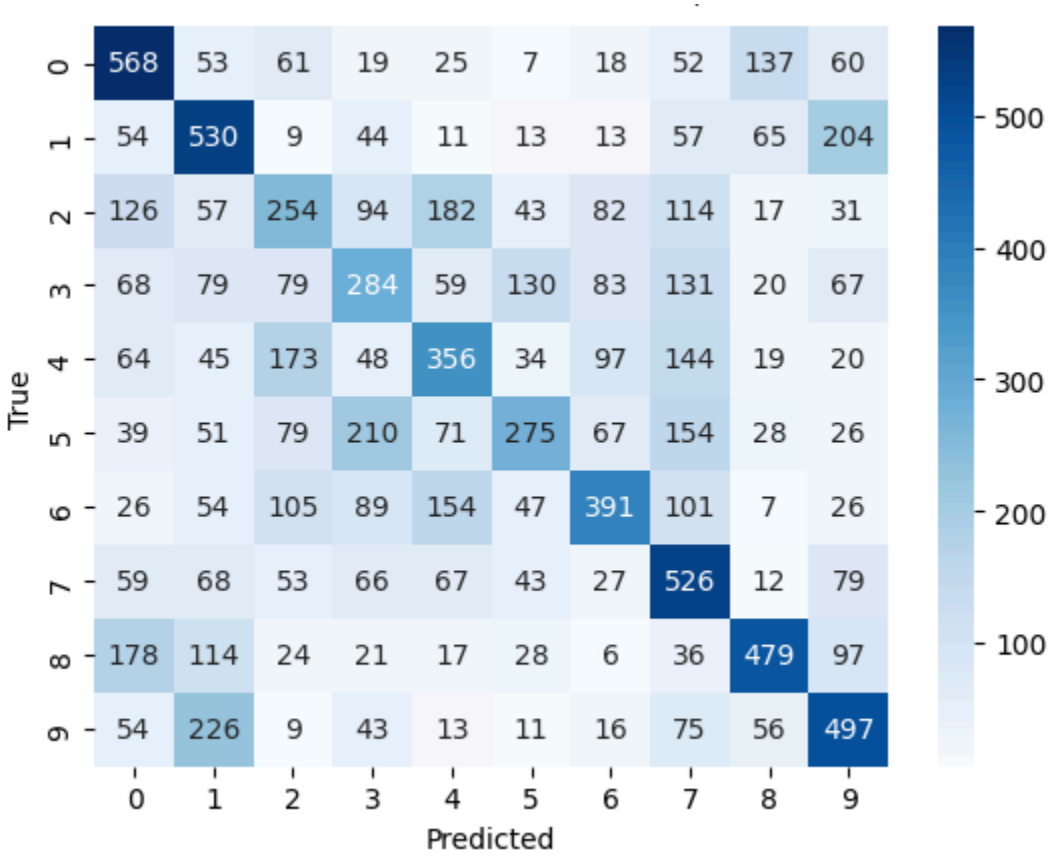
Με 256 νευρώνες, dropout 0.5, και προσθήκη ακόμα ενός layer με 64 νευρώνες: 40.44% με 40 λεπτά ανά fold



Προτιμήθηκε φυσικά η προσθήκη μόνο ενός Dense layer με 256 νευρώνες.

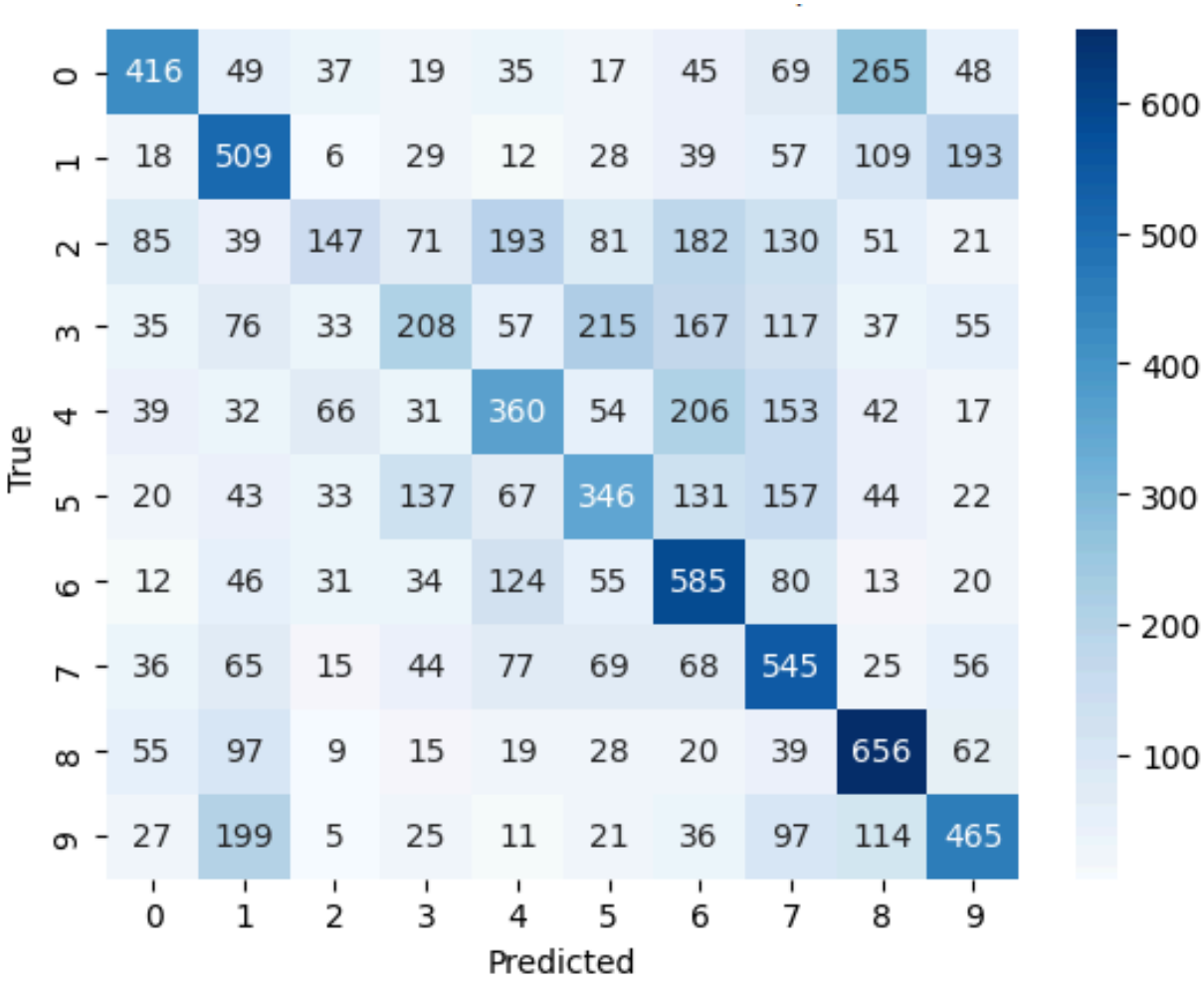
Τέλος, δοκιμάστηκαν 1500 κέντρα με 120 εποχές με 5-fold cross validation και όλες οι επιλογές που τεκμηριώθηκαν προηγουμένως.

41.60% με 60 λεπτά ανά fold. Δυσανάλογη βελτίωση με υπερ-αύξηση του χρόνου εκπαίδευσης. Δεν προτιμήθηκε.



ΤΕΛΙΚΟ ΜΟΝΤΕΛΟ:

42.37% με 150 εποχές ανά fold



Παρουσίαση παραδειγμάτων επιτυχίας και αποτυχίας classification

Correct
True: bird
Predicted: bird

Correct
True: automobile
Predicted: automobile

Correct
True: cat
Predicted: cat

Incorrect
True: truck
Predicted: ship

Incorrect
True: automobile
Predicted: ship

Incorrect
True: cat
Predicted: horse

Σύγκριση με τους κατηγοριοποιητές 1-NN, 3-NN και Nearest Centroid

Στο πρόβλημα κατηγοριοποίησης των εικόνων της βάσης CIFAR-10, έγινε σύγκριση ανάμεσα σε διάφορους κατηγοριοποιητές: το RBF νευρωνικό δίκτυο και τους 1-NN, 3-NN και Nearest Centroid. Το RBF είχε την καλύτερη ακρίβεια (~42%) γιατί είναι πιο «έξυπνο», αλλά χρειάζεται περισσότερο χρόνο για να εκπαιδευτεί λόγω του K-means. Οι 1-NN και 3-NN είναι πολύ απλοί και δε χρειάζονται εκπαίδευση, αλλά είναι αργοί όταν πρέπει να κάνουν προβλέψεις, αφού υπολογίζουν αποστάσεις από όλα τα δείγματα. Ο Nearest Centroid είναι ο πιο γρήγορος και απλός, αλλά δεν είναι πολύ ακριβής γιατί βασίζεται μόνο στους μέσους όρους. Γενικά, το RBF είναι η καλύτερη λύση αν χρειάζεται κάποιος καλή ακρίβεια, ενώ οι άλλοι είναι πιο κατάλληλοι αν προτιμάς κάτι πιο απλό και γρήγορο.