Ομάδα Δ13

Τεχνολογία και Ανάλυση Εικόνων και Βίντεο

3η ομαδική άσκηση

Γιατί και πώς τα συνελικτικά δίκτυα άλλαξαν το τοπίο της παραδοσιακής επεξεργασίας εικόνας;

Το βασικό προτέρημα των συνελικτικών δικτύων είναι πως απαλλάσσουν από την ανάγκη εξαγωγής συγκεκριμένων χαρακτηριστικών των εικόνων (feature extraction), καθώς αυτά εξάγονται αυτομάτως από τα διάφορα συνελικτικά επίπεδα (convolutional layers) του νευρωνικού δικτύου. Η αλλαγή αυτή είναι σημαντική διότι στο παρελθόν η διαδικασία της επιλογής κατάλληλων για εξαγωγή χαρακτηριστικών έπρεπε να γίνει χειρωνακτικά προτού εκπαιδευθεί το δίκτυο. Επιπλέον, έχουν σημαντικά λιγότερες παραμέτρους οπότε εκπαιδεύονται ευκολότερα, ενώ θεωρητικά η βέλτιστη απόδοσή τους είναι αμελητέως χαμηλότερη.

Σε ένα συνελικτικό δίκτυο ο εντοπισμός χαρακτηριστικών (low – middle – high level) επιτυγχάνεται από την ακολουθία convolutional layer, όπου διάφορα εξειδικευμένα φίλτρα εφαρμόζονται συνελικτικά στην εικόνα, activation function layer (συνήθως ReLU) το οποίο αποφαίνεται για την παρουσία του χαρακτηριστικού, και pooling layer όπου η εικόνα συρρικνώνεται ώστε να είναι ευκολότερα διαχειρίσιμη.

Περιγράψτε τη λειτουργία της Rectified Linear Unit (ReLu).

Η ReLU απαλείφει τις αρνητικές τιμές και εφαρμόζει μη γραμμικό μετασχηματισμό στα δεδομένα, το οποίο είναι απαραίτητο για την εκπαίδευση του CNN. Το πλεονέκτημά της υπέρ της σιγμοειδούς είναι ότι εκπαιδεύει πολύ γρηγορότερα, καθώς η παράγωγος της σιγμοειδούς γίνεται πολύ μικρή στην περιοχή κορεσμού και ως εκ τούτου οι μεταβολές στα βάρη είναι σχεδόν μηδενικές ("vanishing gradient problem"). Στο δίκτυο, ένα επίπεδο ReLu τοποθετείται μετά από κάθε συνελικτικό και FC επίπεδο.

$$ReLU : f(x) = \max(0, x)$$

Παρουσιάστε έναν συγκριτικό πίνακα με τα επίπεδα, το μέγεθος των φίλτρων, συνάρτηση ενεργοποίησης, πλήθος παραμέτρων και τεχνικές pooling και dropout που χρησιμοποιούν τα τρία διαφορετικά δίκτυα που παρουσιάζονται στα προαναφερθέντα άρθρα και σχολιάστε επαρκώς τις διαφορές τους. Ποιά χαρακτηριστικά αυτών των τριών δικτύων είναι κατά τη γνώμη σας αυτά που συνέβαλαν στη σταδιακή αύξηση της ακρίβειας ταξινόμησης;

Network	Layers	Filter Size	Enable Function	Parameters	Image Dimensions	Pooling	Dropout
Lenet-5	5 (+2 Pool)	5x5	Sigmoid	60k	24x24	Average Pool	None
AlexNet	8 (+3 Pool)	11x11, 5x5, 3x3	ReLu	62m	256x256	Max Pool	0.5
VGG-16	19	3x3	ReLu	138m	224x224	Max Pool	0.5

Lenet-5:
Η αρχιτεκτονική του ήταν σχεδιασμένη για την αναγνώριση χειρόγραφων αριθμών στο MNIST data-set. Οι εικόνες που λαμβάνει στην είσοδο έχουν μέγεθος 28x28x1 και ακολουθούν δύο Convolution Layers με Stide 2 και Average Pooling με Stride 2. Στο τέλος ακολουθούν fully connected layers με Sigmoid συνάρτηρη ενεργοποίησης. Αποτελείται από 7 επίπεδα. Οι συνολικές παράμετροι κυμαίνονται περίπου στις 60.000 και δεν έχει Dropout.

AlexNet:
Η αρχιτεκτονική του AlexNet μοιάζει πολύ με του LeNet-5 αλλά αποτελείται από περισσότερα επίπεδα, 8 αντί για 5 και 11 συνολικά μαζί με τα pooling. Επίσης έχει περισσότερα φίλτρα και διαδοχικά Convolutional Layers. Αντί για Average Pooling έχει Max Pooling και επίσης έχει Dropout και συνάρτηση ενεργοποίησης ReLu αντί για Sigmoid. Συνολικά έχει 5 Convolutional Layers και τρία Fully Connected Layers. Αποτελείται από περίπου 60.000.000 παραμέτρους, σημαντική αύξηση συγκριτικά με το Lenet-5. Ενα μειωνέκτημα του AlexNet είναι πως έχει πολλές υβριδικές παραμέτρους.

VGG:
Το VGG κατάφερε να επιλύσει το πρόβλημα του AlexNet με τις πολλές υβριδικές μεταβλητές αντικαθιστώντας τα φίλτρα μεγέθους 11x11 και 5x5 στα πρώτα δύο επίπεδα με πολλαπλά 3x3 φίλτρα. Η αρχιτεκτονική αποτελείται από 3x3 Convolutional φίλτρα και 2x2 Max Pooling Layers με Stride 2, όπως και το AlexNet. Συνολικά έχει 16 επίπεδα και η αρχική εικόνα ακολουθείται από 5 ζευγάρια από Convolution φίλτρα και Max Pooling. Η έξοδος από αυτά τα επίπεδα δίνεται σε τρία Fully Connected Layers. Συνολικά έχει 138.000.000 παραμέτρους. Η συνάρτηση ενεργοποίησης και το Dropout είναι ίδια με το AlexNet. Τα μειωνεκτήματα του VGG είναι πως χρειάζεται πολύ χρόνο για training και είναι πολύ βαρύ μοντέλο και υπολογιστικά βαρύ.

Στην αύξηση της ακρίβειας ταξινόμισης φαίνεται να συνέβαλε η χρήση πολλών επιπέδων και μάλιστα πολλών διαδοχικών φίλτρων μικρού μεγέθους ακολουθούμενα από Max Pooling επίπεδα. Το dropout που χρησιμοποιείται αρκετά στις νεότερες μεθόδους βοηθάει πολύ για τον περιορισμό της υπερεκπαίδευσης. Επίσης η αύξηση των συνολικών παραμέτρων βοηθάει πολύ την ακρίβεια αλλά απαιτεί υπολογιστικούς πόρους, οπότε καλούμαστε να επιλέξουμε ανάμεσα στο υπολογιστικό κόστος και την ακρίβεια.

Εισαγωγή και επισκόπηση του συνόλου δεδομένων

In [ ]:
```
from __future__ import absolute_import, division, print_function, unicode_literals # legacy compatibility

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [ ]:
```
# helper functions

# select from from_list elements with index in index_list
def select_from_list(from_list, index_list):
    filtered_list= [from_list[i] for i in index_list]
    return(filtered_list)

# append in filtered_list the index of each element of unfilterd_list if it exists in in target_list
def get_ds_index(unfiltered_list, target_list):
    index = 0
    filtered_list=[]
    for i_ in unfiltered_list:
        if i_[0] in target_list:
            filtered_list.append(index)
        index += 1
    return(filtered_list)
```

In [ ]:
```
!pip3 install tensorflow
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.7/dist-packages (2.5.0)
Requirement already satisfied: astunparse~=1.6.3 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: protobuf>=3.9.2 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (3.17.3)
Requirement already satisfied: h5py~=3.1.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (3.1.0)
Requirement already satisfied: termcolor~=1.1.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.1.0)
Requirement already satisfied: absl-py~=0.10 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (0.12.0)
Requirement already satisfied: tensorflow-estimator<2.6.0,>=2.5.0rc0 in /usr/local/lib/python3.7/dist-packages (from ten
sorflow) (2.5.0)
Requirement already satisfied: grpcio~=1.34.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.34.1)
Requirement already satisfied: keras-preprocessing~=1.1.2 in /usr/local/lib/python3.7/dist-packages (from tensorflow)
(1.1.2)
Requirement already satisfied: tensorboard~=2.5 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (2.5.0)
Requirement already satisfied: typing-extensions~=3.7.4 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (3.
7.4.3)
Requirement already satisfied: wheel~=0.35 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (0.36.2)
Requirement already satisfied: numpy~=1.19.2 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.19.5)
Requirement already satisfied: six~=1.15.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.15.0)
Requirement already satisfied: keras-nightly~=2.5.0.dev in /usr/local/lib/python3.7/dist-packages (from tensorflow) (2.
5.0.dev2021032900)
Requirement already satisfied: google-pasta~=0.2 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: wrapt~=1.12.1 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.12.1)
Requirement already satisfied: gast==0.4.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (0.4.0)
Requirement already satisfied: flatbuffers~=1.12.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.12)
Requirement already satisfied: opt-einsum~=3.3.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (3.3.0)
Requirement already satisfied: cached-property; python_version < "3.8" in /usr/local/lib/python3.7/dist-packages (from h
5py~=3.1.0->tensorflow) (1.5.2)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.7/dist-packages (from tensorboard~=2.5->ten
sorflow) (2.23.0)
Requirement already satisfied: google-auth<2,>=1.6.3 in /usr/local/lib/python3.7/dist-packages (from tensorboard~=2.5->t
ensorflow) (1.32.1)
Requirement already satisfied: tensorboard-data-server<0.7.0,>=0.6.0 in /usr/local/lib/python3.7/dist-packages (from ten
sorboard~=2.5->tensorflow) (0.6.1)
Requirement already satisfied: werkzeug>=0.11.15 in /usr/local/lib/python3.7/dist-packages (from tensorboard~=2.5->tenso
rflow) (1.0.1)
Requirement already satisfied: setuptools>=41.0.0 in /usr/local/lib/python3.7/dist-packages (from tensorboard~=2.5->tens
orflow) (57.0.0)
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /usr/local/lib/python3.7/dist-packages (from tensorbo
ard~=2.5->tensorflow) (0.4.4)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.7/dist-packages (from tensorboard~=2.5->tensorf
low) (3.3.4)
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/lib/python3.7/dist-packages (from tensorboard
~=2.5->tensorflow) (1.8.0)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests<3,>=2.21.0->te
nsorboard~=2.5->tensorflow) (3.0.4)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from r
equests<3,>=2.21.0->tensorboard~=2.5->tensorflow) (1.24.3)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests<3,>=2.21.0->tensorb
oard~=2.5->tensorflow) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests<3,>=2.21.0->t
ensorboard~=2.5->tensorflow) (2021.5.30)
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.7/dist-packages (from google-auth<2,>=1.
6.3->tensorboard~=2.5->tensorflow) (0.2.8)
Requirement already satisfied: cachetools<5.0,>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from google-auth<2,>=1.
6.3->tensorboard~=2.5->tensorflow) (4.2.2)
Requirement already satisfied: rsa<5,>=3.1.4; python_version >= "3.6" in /usr/local/lib/python3.7/dist-packages (from go
ogle-auth<2,>=1.6.3->tensorboard~=2.5->tensorflow) (4.7.2)
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.7/dist-packages (from google-auth-oaut
hlib<0.5,>=0.4.1->tensorboard~=2.5->tensorflow) (1.3.0)
Requirement already satisfied: importlib-metadata; python_version < "3.8" in /usr/local/lib/python3.7/dist-packages (fro
m markdown>=2.6.8->tensorboard~=2.5->tensorflow) (4.6.0)
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /usr/local/lib/python3.7/dist-packages (from pyasn1-modules>=0.2.
1->google-auth<2,>=1.6.3->tensorboard~=2.5->tensorflow) (0.4.8)
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.7/dist-packages (from requests-oauthlib>=0.7.0-
>google-auth-oauthlib<0.5,>=0.4.1->tensorboard~=2.5->tensorflow) (3.1.1)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (from importlib-metadata; python_vers
ion < "3.8"->markdown>=2.6.8->tensorboard~=2.5->tensorflow) (3.4.1)
```

In [ ]:
```
# load the entire dataset
import tensorflow as tf
from tensorflow.keras import datasets, layers, models

(x_train_all, y_train_all), (x_test_all, y_test_all) = tf.keras.datasets.cifar100.load_data(label_mode='fine')
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-100-python.tar.gz
169009152/169001437 [==============================] - 2s 0us/step
```

In [ ]:
```
print(x_train_all.shape)
```

```
(50000, 32, 32, 3)
```

Η κάθε ομάδα θα δουλέψει με διαφορετικό υποσύνολο του dataset. Στο επόμενο κελί, αντικαταστήστε την τιμή της μεταβλητής `team_seed` με τον αριθμό που αντιστοιχεί στην ομάδας σας.

In [ ]:
```
# REPLACE WITH YOUR TEAM NUMBER
team_seed = 13
```

In [ ]:
```
# select from CIFAR100 20 classes
cifar100_classes_url = "https://pastebin.com/raw/nzE1n98V"
```

Δημιουργούμε το μοναδικό dataset της ομάδας μας:

In [ ]:
```
team_classes = pd.read_csv(cifar100_classes_url, sep=',', header=None)
CIFAR100_LABELS_LIST = pd.read_csv('https://pastebin.com/raw/qgDaNggt', sep=',', header=None).astype(str).values.tolist(

# our_indices = team_classes.iloc[team_seed,:].values.tolist()
our_indices = [i for i in range(0, 20)]
our_classes = select_from_list(CIFAR100_LABELS_LIST, our_indices)
train_index = get_ds_index(y_train_all, our_indices)
test_index = get_ds_index(y_test_all, our_indices)

x_train_ds = np.asarray(select_from_list(x_train_all, train_index))
y_train_ds = np.asarray(select_from_list(y_train_all, train_index))
x_test_ds = np.asarray(select_from_list(x_test_all, test_index))
y_test_ds = np.asarray(select_from_list(y_test_all, test_index))
```

In [ ]:
```
# print our classes
print(our_classes)
```

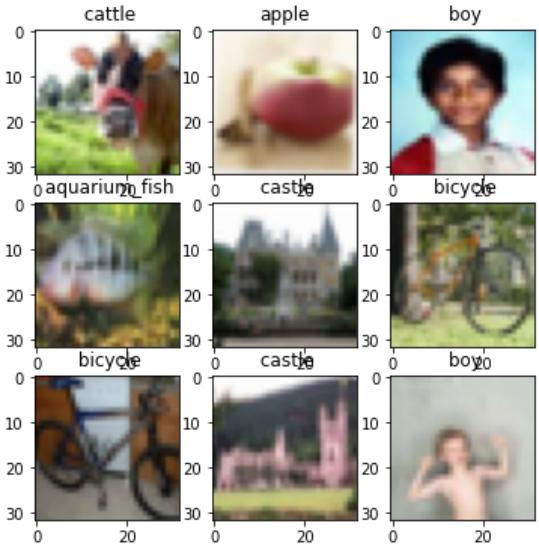
```
['apple', ' aquarium_fish', ' baby', ' bear', ' beaver', ' bed', ' bee', ' beetle', ' bicycle', ' bottle', ' bowl', ' bo
y', ' bridge', ' bus', ' butterfly', ' camel', ' can', ' castle', ' caterpillar', ' cattle']
```

In [ ]:
```
print(x_train_ds[1].shape)
```

```
(32, 32, 3)
```

In [ ]:
```
# get (train) dataset dimensions
data_size, img_rows, img_cols, img_channels = x_train_ds.shape

# set validation set percentage (wrt the training set size)
validation_percentage = 0.15
val_size = round(validation_percentage * data_size)

# Reserve val_size samples for validation and normalize all values
x_val = x_train_ds[-val_size:]/255
y_val = y_train_ds[-val_size:]
x_train = x_train_ds[:-val_size]/255
y_train = y_train_ds[:-val_size]
x_test = x_test_ds/255
y_test = y_test_ds

# summarize loaded dataset
print('Train: X=%s, y=%s' % (x_train.shape, y_train.shape))
print('Validation: X=%s, y=%s' % (x_val.shape, y_val.shape))
print('Test: X=%s, y=%s' % (x_test.shape, y_test.shape))

# get class label from class index
def class_label_from_index(fine_category):
    return(CIFAR100_LABELS_LIST[fine_category.item(0)])

# plot first few images
plt.figure(figsize=(6, 6))
for i in range(9):
    # define subplot
    plt.subplot(330 + 1 + i).set_title(class_label_from_index(y_train[i]))
    # plot raw pixel data
    plt.imshow(x_train[i], cmap=plt.get_cmap('gray'))
    #show the figure
plt.show()
```

```
Train: X=(8500, 32, 32, 3), y=(8500, 1)
Validation: X=(1500, 32, 32, 3), y=(1500, 1)
Test: X=(2000, 32, 32, 3), y=(2000, 1)
```



Ερώτημα 1

Βήμα 1: Σχεδίαση, μεταγλώττιση και εκπαίδευση των μοντέλων

1. Σχεδίαστε, μεταγλωττίστε και εκπαιδεύστε τα μοντέλα **LeNet, AlexNet και VGG**, καθώς και ένα δικό σας μοντέλο (ονομάστε το π.χ. **MyCNN**) χρησιμοποιώντας διαφορετικούς συνδυασμούς (τουλάχιστον 2 ανά μοντέλο) για optimizer, loss, batch size και epoch. Για metrics να χρησιμοποιήσετε την ορθότητα (όπως στο Lab9.1).

Βήμα 2: Αξιολόγηση των μοντέλων

1. Για κάθε ένα από τα μοντέλα που εκπαιδεύσατε, απεικονίστε σε κοινό διάγραμμα την ορθότητα εκπαίδευσης και την ορθότητα επικύρωσης στο σύνολο των εποχών, για κάθε διαφορετικό συνδυασμό του βήματος 1 και επιλέξτε αυτό με την καλύτερη ορθότητα από κάθε μοντέλο (ένα από κάθε αρχιτεκτονική, συνολικά 4).
2. Για κάθε ένα από τα μοντέλα με την καλύτερη ορθότητα (συνολικά **4**), απεικονίστε σε κοινό διάγραμμα την ορθότητα εκπαίδευσης και την ορθότητα επικύρωσης στο σύνολο των εποχών.
3. Αξιολογήστε, αναλυτικά, τα αποτελέσματά σας ως προς τα εξής:
   - Επίδραση του πλήθους των δεδομένων/κλάσεων στην απόδοση του μοντέλου
   - Επίδραση του αλγόριθμου βελτιστοποίησης (optimizer)
   - Επίδραση του μεγέθους δέσμης (batch size)

Βήμα 3: Αξιολόγηση ορθότητας

Αξιολογήστε την ορθότητα για το test set σας.

LeNet

In [ ]:
```
import tensorflow as tf
```

In [ ]:
```
model2 = tf.keras.models.Sequential([
        tf.keras.layers.Conv2D(filters=6, kernel_size=5, activation='relu',
                               padding='same', input_shape = (32, 32, 3)),
        tf.keras.layers.AvgPool2D(pool_size=2, strides=2),
        tf.keras.layers.Conv2D(filters=16, kernel_size=5,
                               activation='relu'),
        tf.keras.layers.AvgPool2D(pool_size=2, strides=2),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(120, activation='relu'),
        tf.keras.layers.Dense(84, activation='relu'),
        tf.keras.layers.Dense(20)])
model3 = tf.keras.models.Sequential([
        tf.keras.layers.Conv2D(filters=6, kernel_size=5, activation='relu',
                               padding='same', input_shape = (32, 32, 3)),
```

```
        tf.keras.layers.AvgPool2D(pool_size=2, strides=2),
        tf.keras.layers.Conv2D(filters=16, kernel_size=5,
                               activation='relu'),
        tf.keras.layers.AvgPool2D(pool_size=2, strides=2),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(120, activation='relu'),
        tf.keras.layers.Dense(84, activation='relu'),
        tf.keras.layers.Dense(20)])




model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Conv2D(filters=6, kernel_size=5, activation='relu', padding='same', input_shape = (32, 32, 3))
model.add(tf.keras.layers.AvgPool2D(pool_size=2, strides=2))
model.add(tf.keras.layers.Conv2D(filters=16, kernel_size=5, activation='relu'))
model.add(tf.keras.layers.AvgPool2D(pool_size=2, strides=2))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(120, activation='relu'))
model.add(tf.keras.layers.Dense(84, activation='relu'))
model.add(tf.keras.layers.Dense(20))
## We took a small liberty with the original model, removing the Gaussian activation in the final layer.
## Other than that, this network matches the original LeNet-5 architecture.

model.summary()
model2.summary()
model3.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 32, 32, 6)         456
_________________________________________________________________
average_pooling2d_4 (Average (None, 16, 16, 6)         0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 12, 12, 16)        2416
_________________________________________________________________
average_pooling2d_5 (Average (None, 6, 6, 16)          0
_________________________________________________________________
flatten_2 (Flatten)          (None, 576)               0
_________________________________________________________________
dense_6 (Dense)              (None, 120)               69240
_________________________________________________________________
dense_7 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_8 (Dense)              (None, 20)                1700
=================================================================
Total params: 83,976
Trainable params: 83,976
Non-trainable params: 0
_________________________________________________________________
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 6)         456
_________________________________________________________________
average_pooling2d (AveragePo (None, 16, 16, 6)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 12, 12, 16)        2416
_________________________________________________________________
average_pooling2d_1 (Average (None, 6, 6, 16)          0
_________________________________________________________________
flatten (Flatten)            (None, 576)               0
_________________________________________________________________
dense (Dense)                (None, 120)               69240
_________________________________________________________________
dense_1 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_2 (Dense)              (None, 20)                1700
=================================================================
Total params: 83,976
Trainable params: 83,976
Non-trainable params: 0
_________________________________________________________________
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 32, 32, 6)         456
_________________________________________________________________
average_pooling2d_2 (Average (None, 16, 16, 6)         0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 12, 12, 16)        2416
_________________________________________________________________
average_pooling2d_3 (Average (None, 6, 6, 16)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 576)               0
_________________________________________________________________
dense_3 (Dense)              (None, 120)               69240
_________________________________________________________________
dense_4 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_5 (Dense)              (None, 20)                1700
=================================================================
Total params: 83,976
Trainable params: 83,976
Non-trainable params: 0
_________________________________________________________________
```

Training

In [ ]:

```
## x_train: training images
## y_train: corresponding labels

model.compile(optimizer='adam',
```

```
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history = model.fit(x_train, y_train, epochs=25, batch_size=32,
                    validation_data=(x_val, y_val))


model2.compile(optimizer='sgd',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history2 = model2.fit(x_train, y_train, epochs=25, batch_size=32,
                    validation_data=(x_val, y_val))

model3.compile(optimizer='sgd',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history3 = model3.fit(x_train, y_train, epochs=25, batch_size=128,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/25
266/266 [==============================] - 11s 40ms/step - loss: 2.6885 - accuracy: 0.1718 - val_loss: 2.3403 - val_accu
racy: 0.2987
Epoch 2/25
266/266 [==============================] - 11s 40ms/step - loss: 2.2782 - accuracy: 0.3101 - val_loss: 2.1877 - val_accu
racy: 0.3253
Epoch 3/25
266/266 [==============================] - 11s 40ms/step - loss: 2.1164 - accuracy: 0.3544 - val_loss: 2.2366 - val_accu
racy: 0.3147
Epoch 4/25
266/266 [==============================] - 11s 40ms/step - loss: 1.9872 - accuracy: 0.4002 - val_loss: 2.0387 - val_accu
racy: 0.3780
Epoch 5/25
266/266 [==============================] - 11s 40ms/step - loss: 1.8652 - accuracy: 0.4338 - val_loss: 1.9732 - val_accu
racy: 0.3867
Epoch 6/25
266/266 [==============================] - 11s 40ms/step - loss: 1.7695 - accuracy: 0.4520 - val_loss: 1.9314 - val_accu
racy: 0.4127
Epoch 7/25
266/266 [==============================] - 11s 40ms/step - loss: 1.6714 - accuracy: 0.4853 - val_loss: 1.9190 - val_accu
racy: 0.4207
Epoch 8/25
266/266 [==============================] - 11s 40ms/step - loss: 1.5842 - accuracy: 0.5095 - val_loss: 1.8961 - val_accu
racy: 0.4227
Epoch 9/25
266/266 [==============================] - 11s 40ms/step - loss: 1.5108 - accuracy: 0.5369 - val_loss: 1.9036 - val_accu
racy: 0.4347
Epoch 10/25
266/266 [==============================] - 11s 40ms/step - loss: 1.4280 - accuracy: 0.5562 - val_loss: 1.8794 - val_accu
racy: 0.4487
Epoch 11/25
266/266 [==============================] - 11s 40ms/step - loss: 1.3466 - accuracy: 0.5865 - val_loss: 1.9491 - val_accu
racy: 0.4280
Epoch 12/25
266/266 [==============================] - 11s 40ms/step - loss: 1.2739 - accuracy: 0.6074 - val_loss: 1.9603 - val_accu
racy: 0.4460
Epoch 13/25
266/266 [==============================] - 11s 40ms/step - loss: 1.1925 - accuracy: 0.6301 - val_loss: 1.9408 - val_accu
racy: 0.4653
Epoch 14/25
266/266 [==============================] - 11s 40ms/step - loss: 1.1207 - accuracy: 0.6547 - val_loss: 2.0885 - val_accu
racy: 0.4327
Epoch 15/25
266/266 [==============================] - 11s 41ms/step - loss: 1.0379 - accuracy: 0.6794 - val_loss: 2.0723 - val_accu
racy: 0.4400
Epoch 16/25
266/266 [==============================] - 11s 41ms/step - loss: 0.9769 - accuracy: 0.6927 - val_loss: 2.2831 - val_accu
racy: 0.4367
Epoch 17/25
266/266 [==============================] - 11s 41ms/step - loss: 0.8924 - accuracy: 0.7198 - val_loss: 2.1799 - val_accu
racy: 0.4553
Epoch 18/25
266/266 [==============================] - 11s 40ms/step - loss: 0.8179 - accuracy: 0.7440 - val_loss: 2.2801 - val_accu
racy: 0.4487
Epoch 19/25
266/266 [==============================] - 11s 40ms/step - loss: 0.7716 - accuracy: 0.7566 - val_loss: 2.4450 - val_accu
racy: 0.4413
Epoch 20/25
266/266 [==============================] - 11s 40ms/step - loss: 0.7045 - accuracy: 0.7786 - val_loss: 2.5535 - val_accu
racy: 0.4280
Epoch 21/25
266/266 [==============================] - 11s 41ms/step - loss: 0.6381 - accuracy: 0.7995 - val_loss: 2.5824 - val_accu
racy: 0.4387
Epoch 22/25
266/266 [==============================] - 11s 41ms/step - loss: 0.5718 - accuracy: 0.8242 - val_loss: 2.7113 - val_accu
racy: 0.4493
Epoch 23/25
266/266 [==============================] - 11s 40ms/step - loss: 0.5306 - accuracy: 0.8322 - val_loss: 2.8425 - val_accu
racy: 0.4420
Epoch 24/25
266/266 [==============================] - 11s 40ms/step - loss: 0.4659 - accuracy: 0.8511 - val_loss: 2.8645 - val_accu
racy: 0.4407
Epoch 25/25
266/266 [==============================] - 11s 41ms/step - loss: 0.4389 - accuracy: 0.8608 - val_loss: 3.1906 - val_accu
racy: 0.4433
Epoch 1/25
266/266 [==============================] - 11s 40ms/step - loss: 2.9879 - accuracy: 0.0652 - val_loss: 2.9740 - val_accu
racy: 0.0973
Epoch 2/25
266/266 [==============================] - 11s 40ms/step - loss: 2.9139 - accuracy: 0.1114 - val_loss: 2.8294 - val_accu
```

```
racy: 0.1080
Epoch 3/25
266/266 [==============================] - 11s 40ms/step - loss: 2.7425 - accuracy: 0.1667 - val_loss: 2.6467 - val_accu
racy: 0.1853
Epoch 4/25
266/266 [==============================] - 11s 40ms/step - loss: 2.6140 - accuracy: 0.2055 - val_loss: 2.5331 - val_accu
racy: 0.2340
Epoch 5/25
266/266 [==============================] - 10s 39ms/step - loss: 2.5155 - accuracy: 0.2438 - val_loss: 2.4701 - val_accu
racy: 0.2487
Epoch 6/25
266/266 [==============================] - 11s 39ms/step - loss: 2.4344 - accuracy: 0.2658 - val_loss: 2.4133 - val_accu
racy: 0.2860
Epoch 7/25
266/266 [==============================] - 11s 40ms/step - loss: 2.3615 - accuracy: 0.2855 - val_loss: 2.3987 - val_accu
racy: 0.2667
Epoch 8/25
266/266 [==============================] - 11s 40ms/step - loss: 2.2981 - accuracy: 0.3042 - val_loss: 2.3034 - val_accu
racy: 0.2893
Epoch 9/25
266/266 [==============================] - 11s 40ms/step - loss: 2.2506 - accuracy: 0.3235 - val_loss: 2.2641 - val_accu
racy: 0.3153
Epoch 10/25
266/266 [==============================] - 11s 40ms/step - loss: 2.1987 - accuracy: 0.3389 - val_loss: 2.2623 - val_accu
racy: 0.3067
Epoch 11/25
266/266 [==============================] - 11s 40ms/step - loss: 2.1482 - accuracy: 0.3519 - val_loss: 2.1743 - val_accu
racy: 0.3267
Epoch 12/25
266/266 [==============================] - 11s 40ms/step - loss: 2.1114 - accuracy: 0.3613 - val_loss: 2.1769 - val_accu
racy: 0.3307
Epoch 13/25
266/266 [==============================] - 11s 40ms/step - loss: 2.0753 - accuracy: 0.3754 - val_loss: 2.1200 - val_accu
racy: 0.3480
Epoch 14/25
266/266 [==============================] - 11s 40ms/step - loss: 2.0424 - accuracy: 0.3789 - val_loss: 2.2723 - val_accu
racy: 0.3167
Epoch 15/25
266/266 [==============================] - 11s 40ms/step - loss: 1.9984 - accuracy: 0.3973 - val_loss: 2.0505 - val_accu
racy: 0.3693
Epoch 16/25
266/266 [==============================] - 11s 40ms/step - loss: 1.9616 - accuracy: 0.4051 - val_loss: 2.0601 - val_accu
racy: 0.3627
Epoch 17/25
266/266 [==============================] - 11s 40ms/step - loss: 1.9316 - accuracy: 0.4105 - val_loss: 2.1145 - val_accu
racy: 0.3493
Epoch 18/25
266/266 [==============================] - 11s 40ms/step - loss: 1.8971 - accuracy: 0.4289 - val_loss: 2.1185 - val_accu
racy: 0.3573
Epoch 19/25
266/266 [==============================] - 11s 40ms/step - loss: 1.8638 - accuracy: 0.4349 - val_loss: 2.0639 - val_accu
racy: 0.3647
Epoch 20/25
266/266 [==============================] - 11s 40ms/step - loss: 1.8297 - accuracy: 0.4428 - val_loss: 2.0688 - val_accu
racy: 0.3767
Epoch 21/25
266/266 [==============================] - 11s 40ms/step - loss: 1.7953 - accuracy: 0.4514 - val_loss: 2.1338 - val_accu
racy: 0.3553
Epoch 22/25
266/266 [==============================] - 11s 40ms/step - loss: 1.7680 - accuracy: 0.4574 - val_loss: 2.0304 - val_accu
racy: 0.3813
Epoch 23/25
266/266 [==============================] - 11s 40ms/step - loss: 1.7371 - accuracy: 0.4711 - val_loss: 2.0911 - val_accu
racy: 0.3747
Epoch 24/25
266/266 [==============================] - 11s 40ms/step - loss: 1.7021 - accuracy: 0.4695 - val_loss: 2.0302 - val_accu
racy: 0.3987
Epoch 25/25
266/266 [==============================] - 11s 40ms/step - loss: 1.6730 - accuracy: 0.4842 - val_loss: 2.0518 - val_accu
racy: 0.3827
Epoch 1/25
67/67 [==============================] - 10s 142ms/step - loss: 2.9962 - accuracy: 0.0582 - val_loss: 2.9916 - val_accur
acy: 0.0747
Epoch 2/25
67/67 [==============================] - 9s 141ms/step - loss: 2.9901 - accuracy: 0.0728 - val_loss: 2.9870 - val_accura
cy: 0.0800
Epoch 3/25
67/67 [==============================] - 9s 141ms/step - loss: 2.9842 - accuracy: 0.0801 - val_loss: 2.9811 - val_accura
cy: 0.0680
Epoch 4/25
```

```
67/67 [==============================] - 9s 141ms/step - loss: 2.9757 - accuracy: 0.0815 - val_loss: 2.9719 - val_accura
cy: 0.0747
Epoch 5/25
67/67 [==============================] - 9s 141ms/step - loss: 2.9624 - accuracy: 0.0864 - val_loss: 2.9596 - val_accura
cy: 0.0833
Epoch 6/25
67/67 [==============================] - 9s 141ms/step - loss: 2.9400 - accuracy: 0.1049 - val_loss: 2.9315 - val_accura
cy: 0.0993
Epoch 7/25
67/67 [==============================] - 9s 140ms/step - loss: 2.9041 - accuracy: 0.1209 - val_loss: 2.9068 - val_accura
cy: 0.0840
Epoch 8/25
67/67 [==============================] - 9s 141ms/step - loss: 2.8588 - accuracy: 0.1360 - val_loss: 2.8813 - val_accura
cy: 0.1113
Epoch 9/25
67/67 [==============================] - 9s 140ms/step - loss: 2.8126 - accuracy: 0.1473 - val_loss: 2.8252 - val_accura
cy: 0.1447
Epoch 10/25
67/67 [==============================] - 9s 142ms/step - loss: 2.7777 - accuracy: 0.1560 - val_loss: 2.8680 - val_accura
cy: 0.1107
Epoch 11/25
67/67 [==============================] - 9s 141ms/step - loss: 2.7329 - accuracy: 0.1694 - val_loss: 2.7870 - val_accura
cy: 0.1300
Epoch 12/25
67/67 [==============================] - 9s 141ms/step - loss: 2.6965 - accuracy: 0.1845 - val_loss: 2.7159 - val_accura
cy: 0.1820
Epoch 13/25
67/67 [==============================] - 9s 141ms/step - loss: 2.6618 - accuracy: 0.1908 - val_loss: 2.6500 - val_accura
cy: 0.2000
Epoch 14/25
67/67 [==============================] - 9s 140ms/step - loss: 2.6105 - accuracy: 0.2119 - val_loss: 2.6420 - val_accura
cy: 0.2047
Epoch 15/25
67/67 [==============================] - 9s 140ms/step - loss: 2.5943 - accuracy: 0.2100 - val_loss: 2.6662 - val_accura
cy: 0.1767
Epoch 16/25
67/67 [==============================] - 9s 140ms/step - loss: 2.5537 - accuracy: 0.2266 - val_loss: 2.6280 - val_accura
cy: 0.1907
Epoch 17/25
67/67 [==============================] - 9s 140ms/step - loss: 2.5228 - accuracy: 0.2365 - val_loss: 2.5715 - val_accura
cy: 0.2273
Epoch 18/25
67/67 [==============================] - 9s 141ms/step - loss: 2.4958 - accuracy: 0.2412 - val_loss: 2.5638 - val_accura
cy: 0.2187
Epoch 19/25
67/67 [==============================] - 9s 141ms/step - loss: 2.4776 - accuracy: 0.2534 - val_loss: 2.5108 - val_accura
cy: 0.2367
Epoch 20/25
67/67 [==============================] - 9s 141ms/step - loss: 2.4520 - accuracy: 0.2607 - val_loss: 2.4849 - val_accura
cy: 0.2447
Epoch 21/25
67/67 [==============================] - 9s 141ms/step - loss: 2.4345 - accuracy: 0.2641 - val_loss: 2.4908 - val_accura
cy: 0.2480
Epoch 22/25
67/67 [==============================] - 9s 142ms/step - loss: 2.4172 - accuracy: 0.2695 - val_loss: 2.4817 - val_accura
cy: 0.2580
Epoch 23/25
67/67 [==============================] - 9s 141ms/step - loss: 2.3924 - accuracy: 0.2774 - val_loss: 2.5456 - val_accura
cy: 0.2373
Epoch 24/25
67/67 [==============================] - 9s 141ms/step - loss: 2.3834 - accuracy: 0.2835 - val_loss: 2.4329 - val_accura
cy: 0.2573
Epoch 25/25
67/67 [==============================] - 9s 141ms/step - loss: 2.3628 - accuracy: 0.2874 - val_loss: 2.5038 - val_accura
cy: 0.2300
```

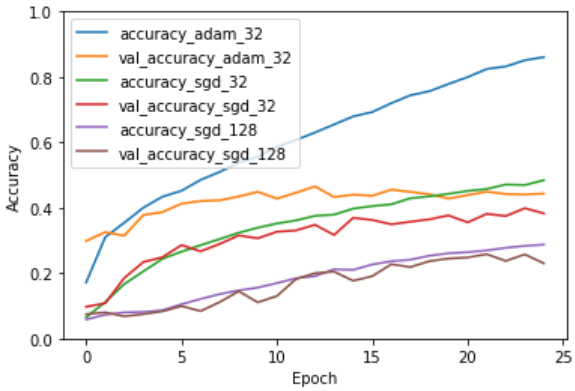
Evaluation

Using too large a batch size can have a negative effect on the accuracy of your network during training since it reduces the stochasticity of the gradient descent.

In [ ]:
```
plt.plot(history.history['accuracy'], label='accuracy_adam_32')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy_adam_32')
plt.plot(history2.history['accuracy'], label='accuracy_sgd_32')
plt.plot(history2.history['val_accuracy'], label = 'val_accuracy_sgd_32')
plt.plot(history3.history['accuracy'], label='accuracy_sgd_128')
plt.plot(history3.history['val_accuracy'], label = 'val_accuracy_sgd_128')

plt.xlabel('Epoch')
```

```
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='upper left')

test_loss, test_acc = model.evaluate(x_test,  y_test, verbose=2)
test_loss2, test_acc2 = model2.evaluate(x_test,  y_test, verbose=2)
test_loss3, test_acc3 = model3.evaluate(x_test,  y_test, verbose=2)
```

```
63/63 - 1s - loss: 3.0510 - accuracy: 0.4425
63/63 - 1s - loss: 2.0169 - accuracy: 0.4040
63/63 - 1s - loss: 2.4607 - accuracy: 0.2430
```



Στο συγκεκριμένο μοντέλο επιλέξαμε να παρουσιάσουμε τρία διαφορετικά υπομοντέλα που διαφέρουν ως προς τον Optimizer αλλά και batch size. Παρατηρούμε πως το batch size έχει σημαντικό ρόλο στην ακρίβεια. Πιο συγκεκριμένα τα batch sizes των 32,64,128 θεωρούνται αξιόλογα και χρειάζεται να πειραματιστεί κανείς με αυτά προτού δημιουργήσει το βέλτισμο μοντέλο. Όσο μεγαλύτερο είναι το batch size τόσο γρηγορότερη είναι και η εκτέλεση του μοντέλου όμως ενδέχεται να έχουμε μικρότερη ακρίβεια όπως φαίνεται στο διάγραμμα για τη σύγκριση των μοντέλων του SGD αλγορίθμου.

Επίσης ο Optimzer παίζει καθοριστικό και ίσως τον σημαντικότερο ρόλο στην ακρίβεια των μοντέλων μας όπως θα φανεί και στη συνέχεια. Στη συγκεκριμένη περίπτωση παρατηρούμε πως η Adam παρουσιάζει σημαντικά καλύτερη εκπαίδευση από το SGD όμως έχει πολύ μεγαλύτερη υπερεκπαίδευση.

Στη συγκεκριμένη περίπτωση θεωρούμε καλύτερο μοντέλο αυτό με Optimizer Adam, batch size 32 και loss function SparseCategoricalCrossentropy.

AlexNet

In [ ]:
```
alex_model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(filters=96, kernel_size=(11,11), strides=(4,4), activation='relu', input_shape=(32,32,3)),
    tf.keras.layers.BatchNormalization(),
    #tf.keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    tf.keras.layers.Conv2D(filters=256, kernel_size=(5,5), strides=(1,1), activation='relu', padding="same"),
    tf.keras.layers.BatchNormalization(),
    #tf.keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    tf.keras.layers.Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), activation='relu', padding="same"),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), activation='relu', padding="same"),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(filters=256, kernel_size=(3,3), strides=(1,1), activation='relu', padding="same"),
    tf.keras.layers.BatchNormalization(),
    #tf.keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(4096, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(4096, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(20, activation='softmax')
])

alex_model2 = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(filters=96, kernel_size=(11,11), strides=(4,4), activation='relu', input_shape=(32,32,3)),
    tf.keras.layers.BatchNormalization(),
    #tf.keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    tf.keras.layers.Conv2D(filters=256, kernel_size=(5,5), strides=(1,1), activation='relu', padding="same"),
```

```
    tf.keras.layers.BatchNormalization(),
    #tf.keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    tf.keras.layers.Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), activation='relu', padding="same"),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), activation='relu', padding="same"),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(filters=256, kernel_size=(3,3), strides=(1,1), activation='relu', padding="same"),
    tf.keras.layers.BatchNormalization(),
    #tf.keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(4096, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(4096, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(20, activation='softmax')
])

alex_model3 = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(filters=96, kernel_size=(11,11), strides=(4,4), activation='relu', input_shape=(32,32,3)),
    tf.keras.layers.BatchNormalization(),
    #tf.keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    tf.keras.layers.Conv2D(filters=256, kernel_size=(5,5), strides=(1,1), activation='relu', padding="same"),
    tf.keras.layers.BatchNormalization(),
    #tf.keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    tf.keras.layers.Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), activation='relu', padding="same"),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), activation='relu', padding="same"),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(filters=256, kernel_size=(3,3), strides=(1,1), activation='relu', padding="same"),
    tf.keras.layers.BatchNormalization(),
    #tf.keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(4096, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(4096, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(20, activation='softmax')
])
```

Training

In [ ]:

```
alex_model.compile(optimizer='adagrad',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
              metrics=['accuracy'])

alex_history = alex_model.fit(x_train, y_train, epochs=20, batch_size=32,
                    validation_data=(x_val, y_val))


alex_model2.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
              metrics=['accuracy'])

alex_history2 = alex_model2.fit(x_train, y_train, epochs=20, batch_size=32,
                    validation_data=(x_val, y_val))


alex_model3.compile(optimizer='sgd',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
              metrics=['accuracy'])

alex_history3 = alex_model3.fit(x_train, y_train, epochs=20, batch_size=32,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/20
266/266 [==============================] - 364s 1s/step - loss: 3.1921 - accuracy: 0.1512 - val_loss: 3.0368 - val_accur
acy: 0.0680
Epoch 2/20
266/266 [==============================] - 364s 1s/step - loss: 2.5753 - accuracy: 0.2505 - val_loss: 2.3304 - val_accur
acy: 0.2913
Epoch 3/20
266/266 [==============================] - 361s 1s/step - loss: 2.3200 - accuracy: 0.3052 - val_loss: 2.1241 - val_accur
acy: 0.3473
Epoch 4/20
266/266 [==============================] - 347s 1s/step - loss: 2.1102 - accuracy: 0.3578 - val_loss: 2.0821 - val_accur
acy: 0.3593
Epoch 5/20
266/266 [==============================] - 327s 1s/step - loss: 1.9460 - accuracy: 0.4039 - val_loss: 2.0090 - val_accur
acy: 0.3853
Epoch 6/20
266/266 [==============================] - 324s 1s/step - loss: 1.8075 - accuracy: 0.4440 - val_loss: 1.9927 - val_accur
acy: 0.4000
Epoch 7/20
266/266 [==============================] - 328s 1s/step - loss: 1.6546 - accuracy: 0.4884 - val_loss: 1.9542 - val_accur
acy: 0.3987
Epoch 8/20
266/266 [==============================] - 329s 1s/step - loss: 1.5242 - accuracy: 0.5253 - val_loss: 1.9282 - val_accur
acy: 0.4120
Epoch 9/20
266/266 [==============================] - 326s 1s/step - loss: 1.4126 - accuracy: 0.5631 - val_loss: 1.9113 - val_accur
acy: 0.4187
Epoch 10/20
266/266 [==============================] - 325s 1s/step - loss: 1.2713 - accuracy: 0.6085 - val_loss: 1.8884 - val_accur
acy: 0.4193
Epoch 11/20
266/266 [==============================] - 325s 1s/step - loss: 1.1592 - accuracy: 0.6441 - val_loss: 1.8800 - val_accur
acy: 0.4213
Epoch 12/20
266/266 [==============================] - 324s 1s/step - loss: 1.0442 - accuracy: 0.6800 - val_loss: 1.8658 - val_accur
acy: 0.4433
Epoch 13/20
266/266 [==============================] - 324s 1s/step - loss: 0.9407 - accuracy: 0.7151 - val_loss: 1.8560 - val_accur
acy: 0.4360
Epoch 14/20
266/266 [==============================] - 325s 1s/step - loss: 0.8491 - accuracy: 0.7435 - val_loss: 1.8671 - val_accur
acy: 0.4500
Epoch 15/20
266/266 [==============================] - 325s 1s/step - loss: 0.7401 - accuracy: 0.7855 - val_loss: 1.8865 - val_accur
acy: 0.4553
Epoch 16/20
266/266 [==============================] - 325s 1s/step - loss: 0.6458 - accuracy: 0.8087 - val_loss: 1.8802 - val_accur
acy: 0.4467
Epoch 17/20
266/266 [==============================] - 325s 1s/step - loss: 0.5682 - accuracy: 0.8379 - val_loss: 1.8917 - val_accur
acy: 0.4520
Epoch 18/20
266/266 [==============================] - 326s 1s/step - loss: 0.4932 - accuracy: 0.8588 - val_loss: 1.9142 - val_accur
acy: 0.4613
Epoch 19/20
266/266 [==============================] - 328s 1s/step - loss: 0.4260 - accuracy: 0.8847 - val_loss: 1.9192 - val_accur
acy: 0.4467
Epoch 20/20
266/266 [==============================] - 327s 1s/step - loss: 0.3743 - accuracy: 0.8993 - val_loss: 1.9580 - val_accur
acy: 0.4560
Epoch 1/20
266/266 [==============================] - 322s 1s/step - loss: 3.5584 - accuracy: 0.1120 - val_loss: 2.9153 - val_accur
acy: 0.1073
Epoch 2/20
266/266 [==============================] - 317s 1s/step - loss: 2.9607 - accuracy: 0.0842 - val_loss: 3.0038 - val_accur
acy: 0.0413
Epoch 3/20
266/266 [==============================] - 319s 1s/step - loss: 2.9976 - accuracy: 0.0475 - val_loss: 2.9989 - val_accur
acy: 0.0413
Epoch 4/20
266/266 [==============================] - 319s 1s/step - loss: 2.9964 - accuracy: 0.0505 - val_loss: 2.9987 - val_accur
acy: 0.0387
Epoch 5/20
266/266 [==============================] - 318s 1s/step - loss: 2.9963 - accuracy: 0.0486 - val_loss: 2.9982 - val_accur
acy: 0.0400
Epoch 6/20
266/266 [==============================] - 317s 1s/step - loss: 2.9961 - accuracy: 0.0496 - val_loss: 2.9985 - val_accur
acy: 0.0387
Epoch 7/20
266/266 [==============================] - 318s 1s/step - loss: 2.9963 - accuracy: 0.0439 - val_loss: 2.9986 - val_accur
```

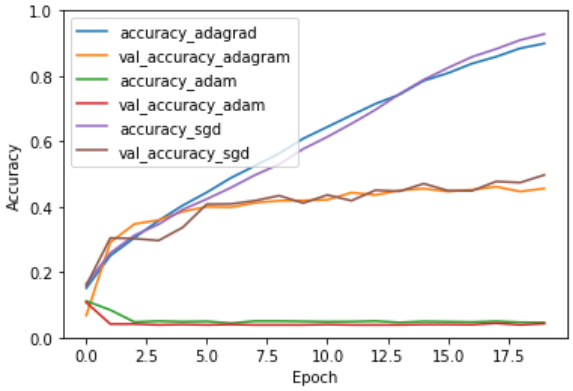
```
acy: 0.0400
Epoch 8/20
266/266 [==============================] - 318s 1s/step - loss: 2.9960 - accuracy: 0.0505 - val_loss: 2.9987 - val_accur
acy: 0.0387
Epoch 9/20
266/266 [==============================] - 317s 1s/step - loss: 2.9959 - accuracy: 0.0505 - val_loss: 2.9988 - val_accur
acy: 0.0387
Epoch 10/20
266/266 [==============================] - 318s 1s/step - loss: 2.9962 - accuracy: 0.0495 - val_loss: 2.9988 - val_accur
acy: 0.0387
Epoch 11/20
266/266 [==============================] - 316s 1s/step - loss: 2.9961 - accuracy: 0.0485 - val_loss: 2.9989 - val_accur
acy: 0.0400
Epoch 12/20
266/266 [==============================] - 318s 1s/step - loss: 2.9961 - accuracy: 0.0488 - val_loss: 2.9988 - val_accur
acy: 0.0387
Epoch 13/20
266/266 [==============================] - 324s 1s/step - loss: 2.9960 - accuracy: 0.0506 - val_loss: 2.9990 - val_accur
acy: 0.0387
Epoch 14/20
266/266 [==============================] - 322s 1s/step - loss: 2.9961 - accuracy: 0.0461 - val_loss: 2.9988 - val_accur
acy: 0.0387
Epoch 15/20
266/266 [==============================] - 318s 1s/step - loss: 2.9960 - accuracy: 0.0495 - val_loss: 2.9987 - val_accur
acy: 0.0400
Epoch 16/20
266/266 [==============================] - 330s 1s/step - loss: 2.9960 - accuracy: 0.0486 - val_loss: 2.9989 - val_accur
acy: 0.0400
Epoch 17/20
266/266 [==============================] - 322s 1s/step - loss: 3.0045 - accuracy: 0.0473 - val_loss: 3.4417 - val_accur
acy: 0.0393
Epoch 18/20
266/266 [==============================] - 321s 1s/step - loss: 3.0032 - accuracy: 0.0499 - val_loss: 3.0007 - val_accur
acy: 0.0440
Epoch 19/20
266/266 [==============================] - 318s 1s/step - loss: 2.9970 - accuracy: 0.0466 - val_loss: 2.9985 - val_accur
acy: 0.0387
Epoch 20/20
266/266 [==============================] - 322s 1s/step - loss: 2.9966 - accuracy: 0.0460 - val_loss: 2.9987 - val_accur
acy: 0.0427
Epoch 1/20
266/266 [==============================] - 307s 1s/step - loss: 3.1923 - accuracy: 0.1640 - val_loss: 2.7003 - val_accur
acy: 0.1593
Epoch 2/20
266/266 [==============================] - 310s 1s/step - loss: 2.5058 - accuracy: 0.2587 - val_loss: 2.2633 - val_accur
acy: 0.3047
Epoch 3/20
266/266 [==============================] - 320s 1s/step - loss: 2.2941 - accuracy: 0.3126 - val_loss: 2.2998 - val_accur
acy: 0.3027
Epoch 4/20
266/266 [==============================] - 310s 1s/step - loss: 2.1586 - accuracy: 0.3464 - val_loss: 2.2593 - val_accur
acy: 0.2967
Epoch 5/20
266/266 [==============================] - 306s 1s/step - loss: 1.9988 - accuracy: 0.3913 - val_loss: 2.1266 - val_accur
acy: 0.3367
Epoch 6/20
266/266 [==============================] - 309s 1s/step - loss: 1.8815 - accuracy: 0.4236 - val_loss: 1.9512 - val_accur
acy: 0.4080
Epoch 7/20
266/266 [==============================] - 303s 1s/step - loss: 1.7538 - accuracy: 0.4584 - val_loss: 1.9592 - val_accur
acy: 0.4087
Epoch 8/20
266/266 [==============================] - 304s 1s/step - loss: 1.6122 - accuracy: 0.4971 - val_loss: 1.8898 - val_accur
acy: 0.4187
Epoch 9/20
266/266 [==============================] - 325s 1s/step - loss: 1.4907 - accuracy: 0.5299 - val_loss: 1.8644 - val_accur
acy: 0.4340
Epoch 10/20
266/266 [==============================] - 324s 1s/step - loss: 1.3339 - accuracy: 0.5768 - val_loss: 1.9593 - val_accur
acy: 0.4113
Epoch 11/20
266/266 [==============================] - 327s 1s/step - loss: 1.2226 - accuracy: 0.6138 - val_loss: 1.8969 - val_accur
acy: 0.4360
Epoch 12/20
266/266 [==============================] - 325s 1s/step - loss: 1.1070 - accuracy: 0.6541 - val_loss: 1.9740 - val_accur
acy: 0.4187
Epoch 13/20
266/266 [==============================] - 325s 1s/step - loss: 0.9389 - accuracy: 0.6968 - val_loss: 1.9502 - val_accur
acy: 0.4507
Epoch 14/20
```

```
266/266 [==============================] - 324s 1s/step - loss: 0.8115 - accuracy: 0.7453 - val_loss: 1.9243 - val_accur
acy: 0.4473
Epoch 15/20
266/266 [==============================] - 323s 1s/step - loss: 0.6627 - accuracy: 0.7887 - val_loss: 1.9228 - val_accur
acy: 0.4707
Epoch 16/20
266/266 [==============================] - 303s 1s/step - loss: 0.5521 - accuracy: 0.8248 - val_loss: 2.0570 - val_accur
acy: 0.4493
Epoch 17/20
266/266 [==============================] - 306s 1s/step - loss: 0.4399 - accuracy: 0.8584 - val_loss: 2.2183 - val_accur
acy: 0.4487
Epoch 18/20
266/266 [==============================] - 300s 1s/step - loss: 0.3632 - accuracy: 0.8827 - val_loss: 2.1170 - val_accur
acy: 0.4773
Epoch 19/20
266/266 [==============================] - 299s 1s/step - loss: 0.2853 - accuracy: 0.9098 - val_loss: 2.3058 - val_accur
acy: 0.4740
Epoch 20/20
266/266 [==============================] - 296s 1s/step - loss: 0.2358 - accuracy: 0.9284 - val_loss: 2.2307 - val_accur
acy: 0.4973
```

Evaluation

In [ ]:
```
plt.plot(alex_history.history['accuracy'], label='accuracy_adagrad')
plt.plot(alex_history.history['val_accuracy'], label = 'val_accuracy_adagram')
plt.plot(alex_history2.history['accuracy'], label='accuracy_adam')
plt.plot(alex_history2.history['val_accuracy'], label = 'val_accuracy_adam')
plt.plot(alex_history3.history['accuracy'], label='accuracy_sgd')
plt.plot(alex_history3.history['val_accuracy'], label = 'val_accuracy_sgd')

plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='upper left')


alex_test_loss, alex_test_acc = alex_model.evaluate(x_test,  y_test, verbose=2)
alex_test_loss2, alex_test_acc2 = alex_model2.evaluate(x_test,  y_test, verbose=2)
alex_test_loss3, alex_test_acc3 = alex_model3.evaluate(x_test,  y_test, verbose=2)
```

```
63/63 - 16s - loss: 1.9470 - accuracy: 0.4625
63/63 - 16s - loss: 2.9954 - accuracy: 0.0520
63/63 - 15s - loss: 2.1648 - accuracy: 0.4885
```



Στη συγκεκριμένη περίπτωση εξετάζουμε παραλλαγές του μοντέλου AlexNet με διαφορετικούς αλγορίθμους Optimizer. Παρατηρήσαμε πολύ μικρό Accuracy σε κάποιες περιπτώσεις της τάξεως του 5% που υποδεικνύει τυχαία επιλογή, δηλαδή το μοντέλο δεν μαθαίνει. Αυτό φαίνεται από το μοντέλο με τη χρήση του Adam αλογρίθμου όπως φαίνεται στο διάγραμμα. Στη συνέχεια εξετάζοντας άλλα μοντέλα όπως το Adagram και SGD, παρουσίασαν αντίστοιχη συμπεριφορά με έντονη υπερεκπαίδευση και στα δύο αλλά πολύ καλύτερο Accuracy από το Adam.

Συνεπώς το καλύτερο μοντέλο φαίνεται να είναι αυτό που χρησιμοποιεί τον αλγόριθμο SGD για optimizer με loss function SparseCategoricalCrossentropy και batch size 32.

Προφανώς μπορεί να υπάρχουν καλύτερα μοντέλα με πολύ καλύτερη ακρίβεια αλλά δεν δοκιμάσαμε όλους τους πιθανούς συνδυασμούς.

VGG

In [ ]:

```
vggmodel = tf.keras.models.Sequential([
    #1st block
    tf.keras.layers.Conv2D(filters =64, kernel_size =3, padding ='same', activation='relu', input_shape=(32,32,3)),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(filters =64, kernel_size =3, padding ='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    #2nd block
    tf.keras.layers.Conv2D(filters =128, kernel_size =3, padding ='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(filters =128, kernel_size =3, padding ='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    #3rd block
    tf.keras.layers.Conv2D(filters =256, kernel_size =3, padding ='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(filters =256, kernel_size =3, padding ='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(filters =256, kernel_size =3, padding ='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    #tf.keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    #4th block
    tf.keras.layers.Conv2D(filters =512, kernel_size =3, padding ='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(filters =512, kernel_size =3, padding ='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(filters =512, kernel_size =3, padding ='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    #tf.keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    #5th block
    tf.keras.layers.Conv2D(filters =512, kernel_size =3, padding ='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(filters =512, kernel_size =3, padding ='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(filters =512, kernel_size =3, padding ='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    #tf.keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(4096, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(4096, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(20, activation='softmax')
])

vggmodel2 = tf.keras.models.Sequential([
    #1st block
    tf.keras.layers.Conv2D(filters =64, kernel_size =3, padding ='same', activation='relu', input_shape=(32,32,3)),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(filters =64, kernel_size =3, padding ='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    #2nd block
    tf.keras.layers.Conv2D(filters =128, kernel_size =3, padding ='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(filters =128, kernel_size =3, padding ='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    #3rd block
    tf.keras.layers.Conv2D(filters =256, kernel_size =3, padding ='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(filters =256, kernel_size =3, padding ='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(filters =256, kernel_size =3, padding ='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    #tf.keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    #4th block
    tf.keras.layers.Conv2D(filters =512, kernel_size =3, padding ='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(filters =512, kernel_size =3, padding ='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(filters =512, kernel_size =3, padding ='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    #tf.keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    #5th block
    tf.keras.layers.Conv2D(filters =512, kernel_size =3, padding ='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(filters =512, kernel_size =3, padding ='same', activation='relu'),
```

```
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(filters =512, kernel_size =3, padding ='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    #tf.keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(4096, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(4096, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(20, activation='softmax')
])

vggmodel3 = tf.keras.models.Sequential([
    #1st block
    tf.keras.layers.Conv2D(filters =64, kernel_size =3, padding ='same', activation='relu', input_shape=(32,32,3)),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(filters =64, kernel_size =3, padding ='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    #2nd block
    tf.keras.layers.Conv2D(filters =128, kernel_size =3, padding ='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(filters =128, kernel_size =3, padding ='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    #3rd block
    tf.keras.layers.Conv2D(filters =256, kernel_size =3, padding ='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(filters =256, kernel_size =3, padding ='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(filters =256, kernel_size =3, padding ='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    #tf.keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    #4th block
    tf.keras.layers.Conv2D(filters =512, kernel_size =3, padding ='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(filters =512, kernel_size =3, padding ='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(filters =512, kernel_size =3, padding ='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    #tf.keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    #5th block
    tf.keras.layers.Conv2D(filters =512, kernel_size =3, padding ='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(filters =512, kernel_size =3, padding ='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(filters =512, kernel_size =3, padding ='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    #tf.keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(4096, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(4096, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(20, activation='softmax')
])
```

Training

In [ ]:
```
vggmodel.compile(optimizer='adagrad', loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True), metrics=['ac

vgghistory = vggmodel.fit(x_train, y_train, epochs=8, batch_size=64, validation_data=(x_val, y_val))

vggmodel2.compile(optimizer='adagrad', loss=tf.keras.losses.MeanAbsoluteError(), metrics=['accuracy'])

vgghistory2 = vggmodel2.fit(x_train, y_train, epochs=8, batch_size=128, validation_data=(x_val, y_val))

vggmodel3.compile(optimizer='sgd', loss=tf.keras.losses.Poisson(reduction = "auto"), metrics=['accuracy'])

vgghistory3 = vggmodel3.fit(x_train, y_train, epochs=8, batch_size=64, validation_data=(x_val, y_val))
```

```
Epoch 1/8
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/backend.py:4930: UserWarning: "`sparse_categorical_crosse
ntropy` received `from_logits=True`, but the `output` argument was produced by a sigmoid or softmax activation and thus
does not represent logits. Was this intended?"
  '"`sparse_categorical_crossentropy` received `from_logits=True`, but '
```

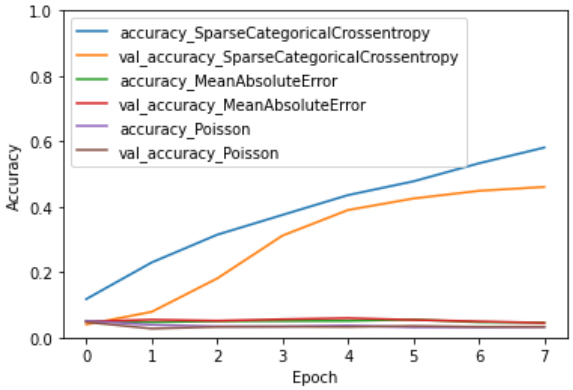
```
133/133 [==============================] - 1266s 10s/step - loss: 3.4462 - accuracy: 0.1176 - val_loss: 3.0898 - val_acc
uracy: 0.0400
Epoch 2/8
133/133 [==============================] - 1243s 9s/step - loss: 2.6971 - accuracy: 0.2296 - val_loss: 3.1061 - val_accu
racy: 0.0787
Epoch 3/8
133/133 [==============================] - 1249s 9s/step - loss: 2.2948 - accuracy: 0.3145 - val_loss: 2.7664 - val_accu
racy: 0.1807
Epoch 4/8
133/133 [==============================] - 1258s 9s/step - loss: 2.0730 - accuracy: 0.3752 - val_loss: 2.2421 - val_accu
racy: 0.3120
Epoch 5/8
133/133 [==============================] - 1254s 9s/step - loss: 1.8560 - accuracy: 0.4356 - val_loss: 2.0005 - val_accu
racy: 0.3900
Epoch 6/8
133/133 [==============================] - 1295s 10s/step - loss: 1.6924 - accuracy: 0.4778 - val_loss: 1.8757 - val_acc
uracy: 0.4253
Epoch 7/8
133/133 [==============================] - 1274s 10s/step - loss: 1.4982 - accuracy: 0.5328 - val_loss: 1.8345 - val_acc
uracy: 0.4487
Epoch 8/8
133/133 [==============================] - 1306s 10s/step - loss: 1.3421 - accuracy: 0.5811 - val_loss: 1.8024 - val_acc
uracy: 0.4607
Epoch 1/8
67/67 [==============================] - 1359s 20s/step - loss: 9.4927 - accuracy: 0.0508 - val_loss: 9.2415 - val_accur
acy: 0.0487
Epoch 2/8
67/67 [==============================] - 1323s 20s/step - loss: 9.4927 - accuracy: 0.0467 - val_loss: 9.2415 - val_accur
acy: 0.0547
Epoch 3/8
67/67 [==============================] - 1321s 20s/step - loss: 9.4927 - accuracy: 0.0500 - val_loss: 9.2415 - val_accur
acy: 0.0513
Epoch 4/8
67/67 [==============================] - 1394s 21s/step - loss: 9.4927 - accuracy: 0.0504 - val_loss: 9.2415 - val_accur
acy: 0.0560
Epoch 5/8
67/67 [==============================] - 1408s 21s/step - loss: 9.4927 - accuracy: 0.0509 - val_loss: 9.2415 - val_accur
acy: 0.0593
Epoch 6/8
67/67 [==============================] - 1490s 22s/step - loss: 9.4927 - accuracy: 0.0551 - val_loss: 9.2415 - val_accur
acy: 0.0540
Epoch 7/8
67/67 [==============================] - 1351s 20s/step - loss: 9.4927 - accuracy: 0.0468 - val_loss: 9.2415 - val_accur
acy: 0.0493
Epoch 8/8
67/67 [==============================] - 1375s 21s/step - loss: 9.4927 - accuracy: 0.0461 - val_loss: 9.2415 - val_accur
acy: 0.0447
Epoch 1/8
133/133 [==============================] - 1260s 9s/step - loss: 33.1318 - accuracy: 0.0509 - val_loss: 28.0808 - val_ac
curacy: 0.0473
Epoch 2/8
133/133 [==============================] - 1232s 9s/step - loss: 28.9145 - accuracy: 0.0388 - val_loss: 27.8722 - val_ac
curacy: 0.0273
Epoch 3/8
133/133 [==============================] - 1217s 9s/step - loss: 28.6280 - accuracy: 0.0340 - val_loss: 27.8716 - val_ac
curacy: 0.0327
Epoch 4/8
133/133 [==============================] - 1222s 9s/step - loss: 28.6247 - accuracy: 0.0339 - val_loss: 27.8721 - val_ac
curacy: 0.0333
Epoch 5/8
133/133 [==============================] - 1226s 9s/step - loss: 28.6236 - accuracy: 0.0367 - val_loss: 27.8723 - val_ac
curacy: 0.0327
Epoch 6/8
133/133 [==============================] - 1224s 9s/step - loss: 28.6231 - accuracy: 0.0313 - val_loss: 27.8720 - val_ac
curacy: 0.0353
Epoch 7/8
133/133 [==============================] - 1223s 9s/step - loss: 28.6228 - accuracy: 0.0316 - val_loss: 27.8717 - val_ac
curacy: 0.0327
Epoch 8/8
133/133 [==============================] - 1243s 9s/step - loss: 28.6227 - accuracy: 0.0324 - val_loss: 27.8712 - val_ac
curacy: 0.0327
```

Evaluation

In [ ]:
```
plt.plot(vgghistory.history['accuracy'], label='accuracy_SparseCategoricalCrossentropy')
plt.plot(vgghistory.history['val_accuracy'], label = 'val_accuracy_SparseCategoricalCrossentropy')
plt.plot(vgghistory2.history['accuracy'], label='accuracy_MeanAbsoluteError')
plt.plot(vgghistory2.history['val_accuracy'], label = 'val_accuracy_MeanAbsoluteError')
plt.plot(vgghistory3.history['accuracy'], label='accuracy_Poisson')
```

```
plt.plot(vgghistory3.history['val_accuracy'], label = 'val_accuracy_Poisson')

plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='upper left')


vggtest_loss, vggtest_acc = vggmodel.evaluate(x_test,  y_test, verbose=2)
vggtest_loss2, vggtest_acc2 = vggmodel2.evaluate(x_test,  y_test, verbose=2)
vggtest_loss3, vggtest_acc3 = vggmodel3.evaluate(x_test,  y_test, verbose=2)
```

```
63/63 - 68s - loss: 1.7413 - accuracy: 0.4635
63/63 - 68s - loss: 9.4550 - accuracy: 0.0505
63/63 - 67s - loss: 28.5115 - accuracy: 0.0360
```



Το συγκεκριμένο μοντέλο VGG ήταν πολύ υπολογιστικά βαρή και προκαλούσε πολλά προβλήματα με τη RAM με αποτέλεσμα να πρέπει να τα ξανατρέχουμε πολλές φορές. Δοκιμάσαμε λοιπόν πολλούς διαφορετικούς συνδυασμούς. Παρατηρήσαμε πως για πολύ μεγάλο batch size είχαμε πολύ καλό accuracy του train set αλλά στο validation set ήταν σαν να γινόταν τυχαία επιλογή. Τελικά φαίνεται η καλύτερη επιλογή να επιτυγχάνεται για batch size 64 ενώ για μικρότερο ή μεγαλύτερο batch size δεν είχαμε ικανοποιητικά αποτελέσματα. Οι loss functions δεν φαίνεται να επηρεάζουν πολύ τα αποτελέσματά μας.

Τελικά το καλύτερο αποτέλεσμα προήλθε από τη χρήση του adagrad Optimizer και batch size 64 ενώ για loss function χρησιμοποιήσαμε τη SparseCategoricalCrossentropy.

MyCNN

Πηγή:

Geron, Aurelien. *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly, 2019.

In [ ]:
```
from functools import partial
```

In [ ]:
```
## ResNet-34
DefaultConv2D = partial(tf.keras.layers.Conv2D, kernel_size=3, strides=1, padding="SAME", use_bias=False)
class ResidualUnit(tf.keras.layers.Layer):
    def __init__(self, filters, strides=1, activation="relu", **kwargs):
        super().__init__(**kwargs)
        self.activation = tf.keras.activations.get(activation)
        self.main_layers = [DefaultConv2D(filters, strides=strides), tf.keras.layers.BatchNormalization(), self.activati
        self.skip_layers = []
        if strides > 1:
            self.skip_layers = [DefaultConv2D(filters, kernel_size=1, strides=strides), tf.keras.layers.BatchNormalizati

    def call(self, inputs):
        Z = inputs
        for layer in self.main_layers:
            Z = layer(Z)
        skip_Z = inputs
        for layer in self.skip_layers:
            skip_Z = layer(skip_Z)
        return self.activation(Z + skip_Z)
```

In [ ]:
```
mymodel = tf.keras.models.Sequential()
mymodel.add(DefaultConv2D(64, kernel_size=7, strides=2, input_shape=[224, 224, 3]))
mymodel.add(tf.keras.layers.BatchNormalization())
mymodel.add(tf.keras.layers.Activation("relu"))
mymodel.add(tf.keras.layers.MaxPool2D(pool_size=3, strides=2, padding="SAME"))
prev_filters = 64
```

```
for filters in [64] * 3 + [128] * 4 + [256] * 6 + [512] * 3:
    strides = 1 if filters == prev_filters else 2
    mymodel.add(ResidualUnit(filters, strides=strides))
    prev_filters = filters
mymodel.add(tf.keras.layers.GlobalAvgPool2D())
mymodel.add(tf.keras.layers.Flatten())
mymodel.add(tf.keras.layers.Dense(20, activation="softmax"))

mymodel.summary()

mymodel2 = tf.keras.models.Sequential()
mymodel2.add(DefaultConv2D(64, kernel_size=7, strides=2, input_shape=[224, 224, 3]))
mymodel2.add(tf.keras.layers.BatchNormalization())
mymodel2.add(tf.keras.layers.Activation("relu"))
mymodel2.add(tf.keras.layers.MaxPool2D(pool_size=3, strides=2, padding="SAME"))
prev_filters = 64
for filters in [64] * 3 + [128] * 4 + [256] * 6 + [512] * 3:
    strides = 1 if filters == prev_filters else 2
    mymodel2.add(ResidualUnit(filters, strides=strides))
    prev_filters = filters
mymodel2.add(tf.keras.layers.GlobalAvgPool2D())
mymodel2.add(tf.keras.layers.Flatten())
mymodel2.add(tf.keras.layers.Dense(20, activation="softmax"))

mymodel2.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 112, 112, 64)      9408
_________________________________________________________________
batch_normalization (BatchNo (None, 112, 112, 64)      256
_________________________________________________________________
activation (Activation)      (None, 112, 112, 64)      0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 56, 56, 64)        0
_________________________________________________________________
residual_unit (ResidualUnit) (None, 56, 56, 64)        74240
_________________________________________________________________
residual_unit_1 (ResidualUni (None, 56, 56, 64)        74240
_________________________________________________________________
residual_unit_2 (ResidualUni (None, 56, 56, 64)        74240
_________________________________________________________________
residual_unit_3 (ResidualUni (None, 28, 28, 128)       230912
_________________________________________________________________
residual_unit_4 (ResidualUni (None, 28, 28, 128)       295936
_________________________________________________________________
residual_unit_5 (ResidualUni (None, 28, 28, 128)       295936
_________________________________________________________________
residual_unit_6 (ResidualUni (None, 28, 28, 128)       295936
_________________________________________________________________
residual_unit_7 (ResidualUni (None, 14, 14, 256)       920576
_________________________________________________________________
residual_unit_8 (ResidualUni (None, 14, 14, 256)       1181696
_________________________________________________________________
residual_unit_9 (ResidualUni (None, 14, 14, 256)       1181696
_________________________________________________________________
residual_unit_10 (ResidualUn (None, 14, 14, 256)       1181696
_________________________________________________________________
residual_unit_11 (ResidualUn (None, 14, 14, 256)       1181696
_________________________________________________________________
residual_unit_12 (ResidualUn (None, 14, 14, 256)       1181696
_________________________________________________________________
residual_unit_13 (ResidualUn (None, 7, 7, 512)         3676160
_________________________________________________________________
residual_unit_14 (ResidualUn (None, 7, 7, 512)         4722688
_________________________________________________________________
residual_unit_15 (ResidualUn (None, 7, 7, 512)         4722688
_________________________________________________________________
global_average_pooling2d (Gl (None, 512)               0
_________________________________________________________________
flatten (Flatten)            (None, 512)               0
_________________________________________________________________
dense (Dense)                (None, 20)                10260
=================================================================
Total params: 21,311,956
Trainable params: 21,294,932
Non-trainable params: 17,024
_________________________________________________________________
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_36 (Conv2D)           (None, 112, 112, 64)      9408
_________________________________________________________________
batch_normalization_36 (Batc (None, 112, 112, 64)      256
_________________________________________________________________
activation_1 (Activation)    (None, 112, 112, 64)      0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 56, 56, 64)        0
_________________________________________________________________
residual_unit_16 (ResidualUn (None, 56, 56, 64)        74240
_________________________________________________________________
residual_unit_17 (ResidualUn (None, 56, 56, 64)        74240
_________________________________________________________________
residual_unit_18 (ResidualUn (None, 56, 56, 64)        74240
_________________________________________________________________
residual_unit_19 (ResidualUn (None, 28, 28, 128)       230912
_________________________________________________________________
residual_unit_20 (ResidualUn (None, 28, 28, 128)       295936
_________________________________________________________________
residual_unit_21 (ResidualUn (None, 28, 28, 128)       295936
_________________________________________________________________
residual_unit_22 (ResidualUn (None, 28, 28, 128)       295936
_________________________________________________________________
```

```
residual_unit_23 (ResidualUn (None, 14, 14, 256)       920576
_________________________________________________________________
residual_unit_24 (ResidualUn (None, 14, 14, 256)       1181696
_________________________________________________________________
residual_unit_25 (ResidualUn (None, 14, 14, 256)       1181696
_________________________________________________________________
residual_unit_26 (ResidualUn (None, 14, 14, 256)       1181696
_________________________________________________________________
residual_unit_27 (ResidualUn (None, 14, 14, 256)       1181696
_________________________________________________________________
residual_unit_28 (ResidualUn (None, 14, 14, 256)       1181696
_________________________________________________________________
residual_unit_29 (ResidualUn (None, 7, 7, 512)         3676160
_________________________________________________________________
residual_unit_30 (ResidualUn (None, 7, 7, 512)         4722688
_________________________________________________________________
residual_unit_31 (ResidualUn (None, 7, 7, 512)         4722688
_________________________________________________________________
global_average_pooling2d_1 ( (None, 512)               0
_________________________________________________________________
flatten_1 (Flatten)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 20)                10260
=================================================================
Total params: 21,311,956
Trainable params: 21,294,932
Non-trainable params: 17,024
_________________________________________________________________
```

Training

In [ ]:

```
mymodel.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

my_history = mymodel.fit(x_train, y_train, epochs=10, batch_size=64,
                     validation_data=(x_val, y_val))
mymodel2.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

my_history2 = mymodel2.fit(x_train, y_train, epochs=10, batch_size=128,
                     validation_data=(x_val, y_val))
```

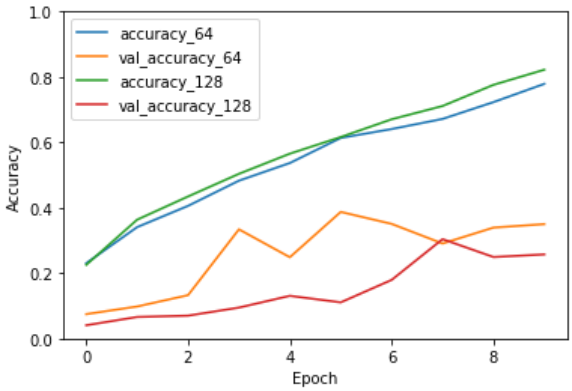
```
Epoch 1/10
WARNING:tensorflow:Model was constructed with shape (None, 224, 224, 3) for input KerasTensor(type_spec=TensorSpec(shape
=(None, 224, 224, 3), dtype=tf.float32, name='conv2d_input'), name='conv2d_input', description="created by layer 'conv2d
_input'"), but it was called on an input with incompatible shape (None, 32, 32, 3).
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/backend.py:4930: UserWarning: "`sparse_categorical_crosse
ntropy` received `from_logits=True`, but the `output` argument was produced by a sigmoid or softmax activation and thus
does not represent logits. Was this intended?"
  '"`sparse_categorical_crossentropy` received `from_logits=True`, but '
```

```
WARNING:tensorflow:Model was constructed with shape (None, 224, 224, 3) for input KerasTensor(type_spec=TensorSpec(shape
=(None, 224, 224, 3), dtype=tf.float32, name='conv2d_input'), name='conv2d_input', description="created by layer 'conv2d
_input'"), but it was called on an input with incompatible shape (None, 32, 32, 3).
133/133 [==============================] - ETA: 0s - loss: 2.6906 - accuracy: 0.2304WARNING:tensorflow:Model was constru
cted with shape (None, 224, 224, 3) for input KerasTensor(type_spec=TensorSpec(shape=(None, 224, 224, 3), dtype=tf.float
32, name='conv2d_input'), name='conv2d_input', description="created by layer 'conv2d_input'"), but it was called on an i
nput with incompatible shape (None, 32, 32, 3).
133/133 [==============================] - 527s 4s/step - loss: 2.6906 - accuracy: 0.2304 - val_loss: 4.2276 - val_accur
acy: 0.0747
Epoch 2/10
133/133 [==============================] - 524s 4s/step - loss: 2.1874 - accuracy: 0.3408 - val_loss: 3.6601 - val_accur
acy: 0.0980
Epoch 3/10
133/133 [==============================] - 529s 4s/step - loss: 1.9701 - accuracy: 0.4058 - val_loss: 3.8607 - val_accur
acy: 0.1327
Epoch 4/10
133/133 [==============================] - 536s 4s/step - loss: 1.6867 - accuracy: 0.4828 - val_loss: 2.3103 - val_accur
acy: 0.3340
Epoch 5/10
133/133 [==============================] - 544s 4s/step - loss: 1.4970 - accuracy: 0.5368 - val_loss: 2.9250 - val_accur
acy: 0.2487
Epoch 6/10
133/133 [==============================] - 545s 4s/step - loss: 1.2658 - accuracy: 0.6135 - val_loss: 2.4349 - val_accur
acy: 0.3873
Epoch 7/10
133/133 [==============================] - 544s 4s/step - loss: 1.1423 - accuracy: 0.6407 - val_loss: 2.6840 - val_accur
acy: 0.3507
Epoch 8/10
133/133 [==============================] - 531s 4s/step - loss: 1.0667 - accuracy: 0.6716 - val_loss: 3.5809 - val_accur
acy: 0.2907
Epoch 9/10
133/133 [==============================] - 530s 4s/step - loss: 0.8846 - accuracy: 0.7231 - val_loss: 2.8742 - val_accur
acy: 0.3393
Epoch 10/10
133/133 [==============================] - 525s 4s/step - loss: 0.6918 - accuracy: 0.7789 - val_loss: 3.0240 - val_accur
acy: 0.3500
Epoch 1/10
WARNING:tensorflow:Model was constructed with shape (None, 224, 224, 3) for input KerasTensor(type_spec=TensorSpec(shape
=(None, 224, 224, 3), dtype=tf.float32, name='conv2d_36_input'), name='conv2d_36_input', description="created by layer
'conv2d_36_input'"), but it was called on an input with incompatible shape (None, 32, 32, 3).
WARNING:tensorflow:Model was constructed with shape (None, 224, 224, 3) for input KerasTensor(type_spec=TensorSpec(shape
=(None, 224, 224, 3), dtype=tf.float32, name='conv2d_36_input'), name='conv2d_36_input', description="created by layer
'conv2d_36_input'"), but it was called on an input with incompatible shape (None, 32, 32, 3).
67/67 [==============================] - ETA: 0s - loss: 2.8031 - accuracy: 0.2254WARNING:tensorflow:Model was construct
ed with shape (None, 224, 224, 3) for input KerasTensor(type_spec=TensorSpec(shape=(None, 224, 224, 3), dtype=tf.float3
2, name='conv2d_36_input'), name='conv2d_36_input', description="created by layer 'conv2d_36_input'"), but it was called
on an input with incompatible shape (None, 32, 32, 3).
67/67 [==============================] - 506s 8s/step - loss: 2.8031 - accuracy: 0.2254 - val_loss: 6.2858 - val_accurac
y: 0.0407
Epoch 2/10
67/67 [==============================] - 503s 8s/step - loss: 2.0899 - accuracy: 0.3635 - val_loss: 3.2733 - val_accurac
y: 0.0660
Epoch 3/10
67/67 [==============================] - 500s 7s/step - loss: 1.8515 - accuracy: 0.4345 - val_loss: 3.2843 - val_accurac
y: 0.0700
Epoch 4/10
67/67 [==============================] - 504s 8s/step - loss: 1.6059 - accuracy: 0.5038 - val_loss: 3.2509 - val_accurac
y: 0.0947
Epoch 5/10
67/67 [==============================] - 507s 8s/step - loss: 1.3985 - accuracy: 0.5654 - val_loss: 3.9405 - val_accurac
y: 0.1307
Epoch 6/10
67/67 [==============================] - 508s 8s/step - loss: 1.2182 - accuracy: 0.6162 - val_loss: 5.4563 - val_accurac
y: 0.1107
Epoch 7/10
67/67 [==============================] - 504s 8s/step - loss: 1.0485 - accuracy: 0.6706 - val_loss: 4.1859 - val_accurac
y: 0.1793
Epoch 8/10
67/67 [==============================] - 506s 8s/step - loss: 0.8986 - accuracy: 0.7112 - val_loss: 3.2030 - val_accurac
y: 0.3040
Epoch 9/10
67/67 [==============================] - 505s 8s/step - loss: 0.6718 - accuracy: 0.7759 - val_loss: 3.9546 - val_accurac
y: 0.2493
Epoch 10/10
67/67 [==============================] - 501s 7s/step - loss: 0.5410 - accuracy: 0.8222 - val_loss: 4.4683 - val_accurac
y: 0.2573
```

Evaluation

In [ ]:
```
plt.plot(my_history.history['accuracy'], label='accuracy_64')
plt.plot(my_history.history['val_accuracy'], label = 'val_accuracy_64')
plt.plot(my_history2.history['accuracy'], label='accuracy_128')
plt.plot(my_history2.history['val_accuracy'], label = 'val_accuracy_128')

plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='upper left')


my_test_loss, my_test_acc = mymodel.evaluate(x_test,  y_test, verbose=2)
my_test_loss2, my_test_acc2 = mymodel2.evaluate(x_test,  y_test, verbose=2)
```

```
63/63 - 6s - loss: 3.0207 - accuracy: 0.3580
63/63 - 6s - loss: 4.4852 - accuracy: 0.2580
```



Στη συγκεκριμένη περίπτωση δημιουργήσαμε ενα δικό μας μοντέλο, χωρίς dropout ή μεθόδους αποφυγής υπερεκπαίδευσης. Επιλέξαμε loss function την SparseCategoricalCrossentropy και Optimizer Adam, ενώ πειραματηστίκαμε με το batch size. Παρατηρούμε πως σε αυτή τη περίπτωση φαίνεται πως και τα δύο batch size να παρουσιάζουν πάρα πολύ καλό Accuracy αλλά σημαντικά χειρότερο Accuracy στο Validation accuracy. Το μεγαλύτερο batch size στη τιμή του 128 είναι αρχικά χειρότερο αλλά στις τελευταίες εποχές βελτιώνεται αρκετά και γίνεται καλύτερο από το batch size 64 και δεδομένου πως είναι ταχύτερο το προτιμήσαμε.

Συνεπώς στο δικό μας CNN επιλέξαμε batch size 128 και τις παραπάνω παραμέτρους σύμφωνα με τις οποίες προχωρήσαμε στα παρακάτω ερωτήματα.

Παρακάτω βλέπετε τα διαγράμματα των Accuracy όσον αφορά τα μοντέλα που επιλέξαμε για κάθε είδος, Lenet, AlexNet, VGG, MYCNN.

In [ ]:
```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history = model.fit(x_train, y_train, epochs=25, batch_size=32,
                    validation_data=(x_val, y_val))

alex_model3.compile(optimizer='sgd',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
              metrics=['accuracy'])

alex_history3 = alex_model3.fit(x_train, y_train, epochs=20, batch_size=32,
                    validation_data=(x_val, y_val))

vggmodel.compile(optimizer='adagrad', loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True), metrics=['ac

vgghistory = vggmodel.fit(x_train, y_train, epochs=8, batch_size=64, validation_data=(x_val, y_val))
```

```
Epoch 1/25
266/266 [==============================] - 12s 43ms/step - loss: 2.6888 - accuracy: 0.1751 - val_loss: 2.3887 - val_accu
racy: 0.2587
Epoch 2/25
266/266 [==============================] - 11s 42ms/step - loss: 2.3059 - accuracy: 0.2921 - val_loss: 2.2915 - val_accu
racy: 0.3013
Epoch 3/25
266/266 [==============================] - 11s 42ms/step - loss: 2.1326 - accuracy: 0.3493 - val_loss: 2.1017 - val_accu
racy: 0.3527
Epoch 4/25
266/266 [==============================] - 11s 42ms/step - loss: 1.9836 - accuracy: 0.3929 - val_loss: 2.0503 - val_accu
racy: 0.3700
Epoch 5/25
266/266 [==============================] - 11s 42ms/step - loss: 1.8622 - accuracy: 0.4325 - val_loss: 1.9428 - val_accu
racy: 0.4020
Epoch 6/25
266/266 [==============================] - 11s 42ms/step - loss: 1.7615 - accuracy: 0.4665 - val_loss: 1.9245 - val_accu
racy: 0.4040
Epoch 7/25
266/266 [==============================] - 11s 42ms/step - loss: 1.6759 - accuracy: 0.4840 - val_loss: 1.8964 - val_accu
racy: 0.4373
Epoch 8/25
266/266 [==============================] - 11s 42ms/step - loss: 1.5851 - accuracy: 0.5115 - val_loss: 1.9164 - val_accu
racy: 0.4220
Epoch 9/25
266/266 [==============================] - 11s 42ms/step - loss: 1.4983 - accuracy: 0.5360 - val_loss: 1.9708 - val_accu
racy: 0.4260
Epoch 10/25
266/266 [==============================] - 11s 42ms/step - loss: 1.4092 - accuracy: 0.5604 - val_loss: 1.9483 - val_accu
racy: 0.4267
Epoch 11/25
266/266 [==============================] - 11s 42ms/step - loss: 1.3372 - accuracy: 0.5842 - val_loss: 1.9383 - val_accu
racy: 0.4380
Epoch 12/25
266/266 [==============================] - 11s 42ms/step - loss: 1.2567 - accuracy: 0.6028 - val_loss: 2.0079 - val_accu
racy: 0.4447
Epoch 13/25
266/266 [==============================] - 11s 43ms/step - loss: 1.1828 - accuracy: 0.6275 - val_loss: 2.0136 - val_accu
racy: 0.4500
Epoch 14/25
266/266 [==============================] - 11s 42ms/step - loss: 1.0978 - accuracy: 0.6496 - val_loss: 2.1681 - val_accu
racy: 0.4447
Epoch 15/25
266/266 [==============================] - 11s 42ms/step - loss: 1.0316 - accuracy: 0.6734 - val_loss: 2.3059 - val_accu
racy: 0.4093
Epoch 16/25
266/266 [==============================] - 11s 42ms/step - loss: 0.9484 - accuracy: 0.7013 - val_loss: 2.2270 - val_accu
racy: 0.4360
Epoch 17/25
266/266 [==============================] - 11s 42ms/step - loss: 0.8876 - accuracy: 0.7195 - val_loss: 2.2980 - val_accu
racy: 0.4460
Epoch 18/25
266/266 [==============================] - 11s 43ms/step - loss: 0.8298 - accuracy: 0.7328 - val_loss: 2.3784 - val_accu
racy: 0.4427
Epoch 19/25
266/266 [==============================] - 11s 43ms/step - loss: 0.7575 - accuracy: 0.7562 - val_loss: 2.4724 - val_accu
racy: 0.4413
Epoch 20/25
266/266 [==============================] - 11s 43ms/step - loss: 0.6841 - accuracy: 0.7875 - val_loss: 2.6559 - val_accu
racy: 0.4367
Epoch 21/25
266/266 [==============================] - 11s 43ms/step - loss: 0.6218 - accuracy: 0.8022 - val_loss: 2.6298 - val_accu
racy: 0.4213
Epoch 22/25
266/266 [==============================] - 11s 43ms/step - loss: 0.5606 - accuracy: 0.8161 - val_loss: 2.8295 - val_accu
racy: 0.4353
Epoch 23/25
266/266 [==============================] - 11s 43ms/step - loss: 0.5263 - accuracy: 0.8287 - val_loss: 2.9751 - val_accu
racy: 0.4340
Epoch 24/25
266/266 [==============================] - 11s 43ms/step - loss: 0.4671 - accuracy: 0.8512 - val_loss: 3.0903 - val_accu
racy: 0.4360
Epoch 25/25
266/266 [==============================] - 12s 43ms/step - loss: 0.4243 - accuracy: 0.8668 - val_loss: 3.2448 - val_accu
racy: 0.4253
Epoch 1/20
266/266 [==============================] - 321s 1s/step - loss: 3.2087 - accuracy: 0.1546 - val_loss: 2.7705 - val_accur
acy: 0.1480
Epoch 2/20
266/266 [==============================] - 320s 1s/step - loss: 2.5404 - accuracy: 0.2526 - val_loss: 2.5135 - val_accur
```

```
acy: 0.2287
Epoch 3/20
266/266 [==============================] - 321s 1s/step - loss: 2.2974 - accuracy: 0.3104 - val_loss: 2.4015 - val_accur
acy: 0.2793
Epoch 4/20
266/266 [==============================] - 324s 1s/step - loss: 2.1328 - accuracy: 0.3532 - val_loss: 2.1595 - val_accur
acy: 0.3387
Epoch 5/20
266/266 [==============================] - 325s 1s/step - loss: 2.0104 - accuracy: 0.3862 - val_loss: 2.1450 - val_accur
acy: 0.3440
Epoch 6/20
266/266 [==============================] - 324s 1s/step - loss: 1.8997 - accuracy: 0.4156 - val_loss: 1.9866 - val_accur
acy: 0.3980
Epoch 7/20
266/266 [==============================] - 322s 1s/step - loss: 1.7719 - accuracy: 0.4560 - val_loss: 2.0513 - val_accur
acy: 0.3827
Epoch 8/20
266/266 [==============================] - 322s 1s/step - loss: 1.6524 - accuracy: 0.4820 - val_loss: 1.9096 - val_accur
acy: 0.4047
Epoch 9/20
266/266 [==============================] - 321s 1s/step - loss: 1.5200 - accuracy: 0.5261 - val_loss: 1.8488 - val_accur
acy: 0.4380
Epoch 10/20
266/266 [==============================] - 321s 1s/step - loss: 1.3918 - accuracy: 0.5600 - val_loss: 1.8385 - val_accur
acy: 0.4367
Epoch 11/20
266/266 [==============================] - 321s 1s/step - loss: 1.2854 - accuracy: 0.5936 - val_loss: 1.8415 - val_accur
acy: 0.4413
Epoch 12/20
266/266 [==============================] - 320s 1s/step - loss: 1.1215 - accuracy: 0.6428 - val_loss: 2.1103 - val_accur
acy: 0.3893
Epoch 13/20
266/266 [==============================] - 319s 1s/step - loss: 0.9914 - accuracy: 0.6780 - val_loss: 1.8965 - val_accur
acy: 0.4367
Epoch 14/20
266/266 [==============================] - 319s 1s/step - loss: 0.8461 - accuracy: 0.7240 - val_loss: 1.8414 - val_accur
acy: 0.4613
Epoch 15/20
266/266 [==============================] - 318s 1s/step - loss: 0.7266 - accuracy: 0.7679 - val_loss: 1.9858 - val_accur
acy: 0.4673
Epoch 16/20
266/266 [==============================] - 318s 1s/step - loss: 0.5784 - accuracy: 0.8176 - val_loss: 2.1721 - val_accur
acy: 0.4520
Epoch 17/20
266/266 [==============================] - 317s 1s/step - loss: 0.4680 - accuracy: 0.8522 - val_loss: 2.0743 - val_accur
acy: 0.4820
Epoch 18/20
266/266 [==============================] - 317s 1s/step - loss: 0.3596 - accuracy: 0.8839 - val_loss: 2.1483 - val_accur
acy: 0.4760
Epoch 19/20
266/266 [==============================] - 316s 1s/step - loss: 0.2996 - accuracy: 0.9049 - val_loss: 2.1264 - val_accur
acy: 0.4853
Epoch 20/20
266/266 [==============================] - 316s 1s/step - loss: 0.2190 - accuracy: 0.9315 - val_loss: 2.2493 - val_accur
acy: 0.4760
Epoch 1/8
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/backend.py:4930: UserWarning: "`sparse_categorical_crosse
ntropy` received `from_logits=True`, but the `output` argument was produced by a sigmoid or softmax activation and thus
does not represent logits. Was this intended?"
  '"`sparse_categorical_crossentropy` received `from_logits=True`, but '
```

```
133/133 [==============================] - 1089s 8s/step - loss: 3.3968 - accuracy: 0.1246 - val_loss: 3.0803 - val_accu
racy: 0.0627
Epoch 2/8
133/133 [==============================] - 1066s 8s/step - loss: 2.6330 - accuracy: 0.2502 - val_loss: 3.3282 - val_accu
racy: 0.0753
Epoch 3/8
133/133 [==============================] - 1052s 8s/step - loss: 2.2952 - accuracy: 0.3245 - val_loss: 3.0412 - val_accu
racy: 0.1133
Epoch 4/8
133/133 [==============================] - 1057s 8s/step - loss: 2.0577 - accuracy: 0.3829 - val_loss: 2.4813 - val_accu
racy: 0.2593
Epoch 5/8
133/133 [==============================] - 1024s 8s/step - loss: 1.8489 - accuracy: 0.4366 - val_loss: 2.0205 - val_accu
racy: 0.3733
Epoch 6/8
133/133 [==============================] - 1000s 8s/step - loss: 1.6621 - accuracy: 0.4895 - val_loss: 1.8717 - val_accu
racy: 0.4387
Epoch 7/8
133/133 [==============================] - 1001s 8s/step - loss: 1.4682 - accuracy: 0.5452 - val_loss: 1.8189 - val_accu
racy: 0.4467
Epoch 8/8
133/133 [==============================] - 997s 7s/step - loss: 1.2923 - accuracy: 0.5973 - val_loss: 1.7760 - val_accur
acy: 0.4727
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-19-f1110131ae1c> in <module>()
     23
     24 my_history2 = mymodel.fit(x_train, y_train, epochs=10, batch_size=128,
---> 25                     validation_data=(x_val, y_val))

/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/training.py in fit(self, x, y, batch_size, epochs,
verbose, callbacks, validation_split, validation_data, shuffle, class_weight, sample_weight, initial_epoch, steps_per_ep
och, validation_steps, validation_batch_size, validation_freq, max_queue_size, workers, use_multiprocessing)
   1103     # Legacy graph support is contained in `training_v1.Model`.
   1104     version_utils.disallow_legacy_graph('Model', 'fit')
-> 1105     self._assert_compile_was_called()
   1106     self._check_call_args('fit')
   1107     _disallow_inside_tf_function('fit')

/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/training.py in _assert_compile_was_called(self)
   2691     # (i.e. whether the model is built and its inputs/outputs are set).
   2692     if not self._is_compiled:
-> 2693       raise RuntimeError('You must compile your model before '
   2694                          'training/testing. '
   2695                          'Use `model.compile(optimizer, loss)`.')

RuntimeError: You must compile your model before training/testing. Use `model.compile(optimizer, loss)`.
```

Το παραπάνω σφάλμα προέκυψε από τυπογραφικό, δε τα ξανατρέξαμε από την αρχή γιατί θέλουν πολύ χρόνο οπότε τρέξαμε ακριβώς από κάτω τη διόρθωση για το myCNN.

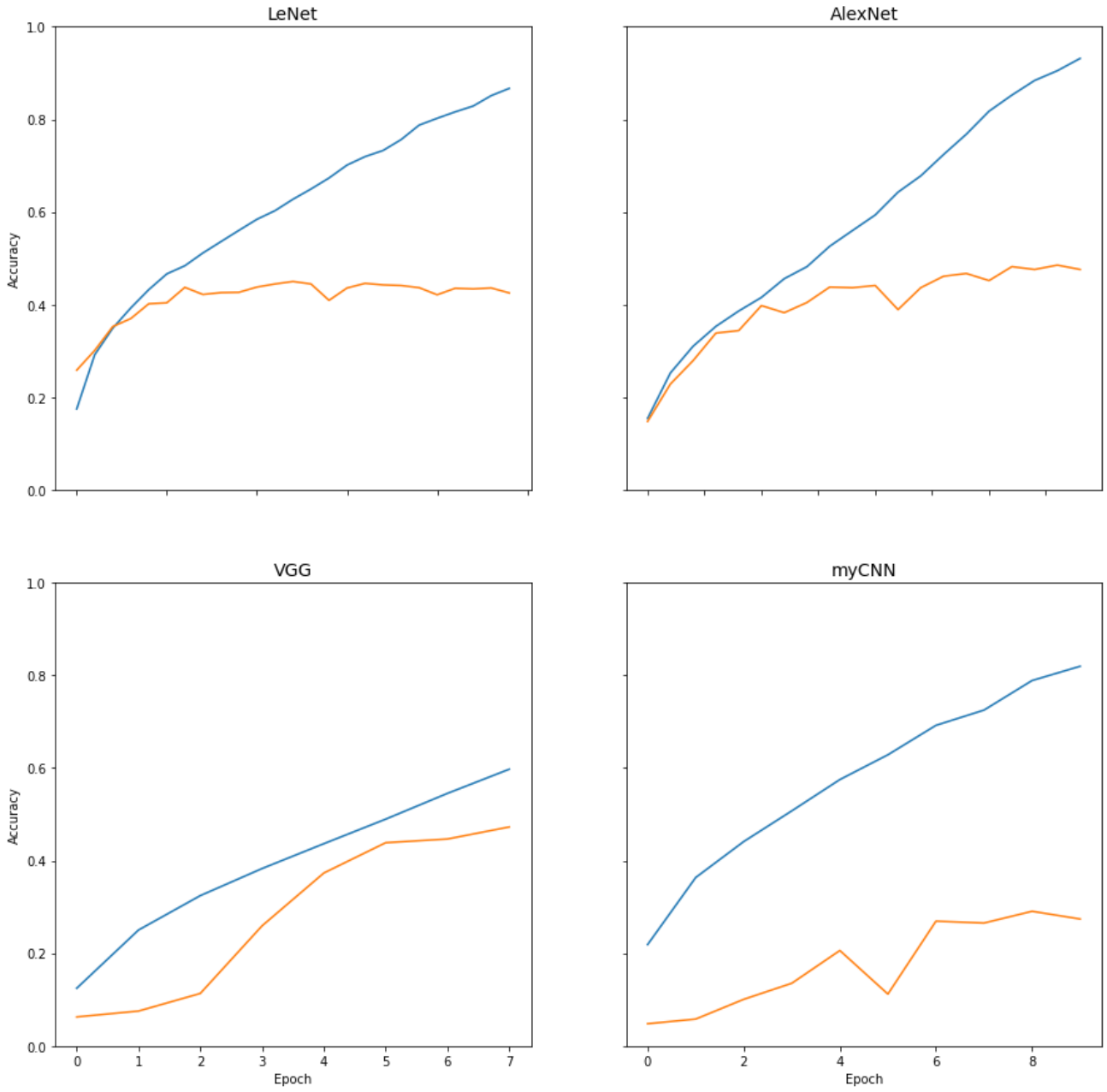
In [ ]:
```
mymodel2.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

my_history2 = mymodel2.fit(x_train, y_train, epochs=10, batch_size=128,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/10
WARNING:tensorflow:Model was constructed with shape (None, 224, 224, 3) for input KerasTensor(type_spec=TensorSpec(shape
=(None, 224, 224, 3), dtype=tf.float32, name='conv2d_96_input'), name='conv2d_96_input', description="created by layer
'conv2d_96_input'"), but it was called on an input with incompatible shape (None, 32, 32, 3).
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/backend.py:4930: UserWarning: "`sparse_categorical_crosse
ntropy` received `from_logits=True`, but the `output` argument was produced by a sigmoid or softmax activation and thus
does not represent logits. Was this intended?"
  '"`sparse_categorical_crossentropy` received `from_logits=True`, but '
```

```
WARNING:tensorflow:Model was constructed with shape (None, 224, 224, 3) for input KerasTensor(type_spec=TensorSpec(shape
=(None, 224, 224, 3), dtype=tf.float32, name='conv2d_96_input'), name='conv2d_96_input', description="created by layer
'conv2d_96_input'"), but it was called on an input with incompatible shape (None, 32, 32, 3).
67/67 [==============================] - ETA: 0s - loss: 2.7391 - accuracy: 0.2188WARNING:tensorflow:Model was construct
ed with shape (None, 224, 224, 3) for input KerasTensor(type_spec=TensorSpec(shape=(None, 224, 224, 3), dtype=tf.float3
2, name='conv2d_96_input'), name='conv2d_96_input', description="created by layer 'conv2d_96_input'"), but it was called
on an input with incompatible shape (None, 32, 32, 3).
67/67 [==============================] - 493s 7s/step - loss: 2.7391 - accuracy: 0.2188 - val_loss: 5.0776 - val_accurac
y: 0.0480
Epoch 2/10
67/67 [==============================] - 484s 7s/step - loss: 2.0952 - accuracy: 0.3635 - val_loss: 3.8477 - val_accurac
y: 0.0580
Epoch 3/10
67/67 [==============================] - 484s 7s/step - loss: 1.8128 - accuracy: 0.4412 - val_loss: 4.2186 - val_accurac
y: 0.1007
Epoch 4/10
67/67 [==============================] - 479s 7s/step - loss: 1.5624 - accuracy: 0.5074 - val_loss: 3.9268 - val_accurac
y: 0.1353
Epoch 5/10
67/67 [==============================] - 477s 7s/step - loss: 1.3600 - accuracy: 0.5749 - val_loss: 3.4453 - val_accurac
y: 0.2060
Epoch 6/10
67/67 [==============================] - 480s 7s/step - loss: 1.1594 - accuracy: 0.6285 - val_loss: 5.8887 - val_accurac
y: 0.1120
Epoch 7/10
67/67 [==============================] - 477s 7s/step - loss: 0.9561 - accuracy: 0.6920 - val_loss: 3.0739 - val_accurac
y: 0.2693
Epoch 8/10
67/67 [==============================] - 479s 7s/step - loss: 0.8531 - accuracy: 0.7249 - val_loss: 3.6874 - val_accurac
y: 0.2653
Epoch 9/10
67/67 [==============================] - 477s 7s/step - loss: 0.6531 - accuracy: 0.7887 - val_loss: 3.4191 - val_accurac
y: 0.2907
Epoch 10/10
67/67 [==============================] - 479s 7s/step - loss: 0.5364 - accuracy: 0.8195 - val_loss: 4.4098 - val_accurac
y: 0.2740
```

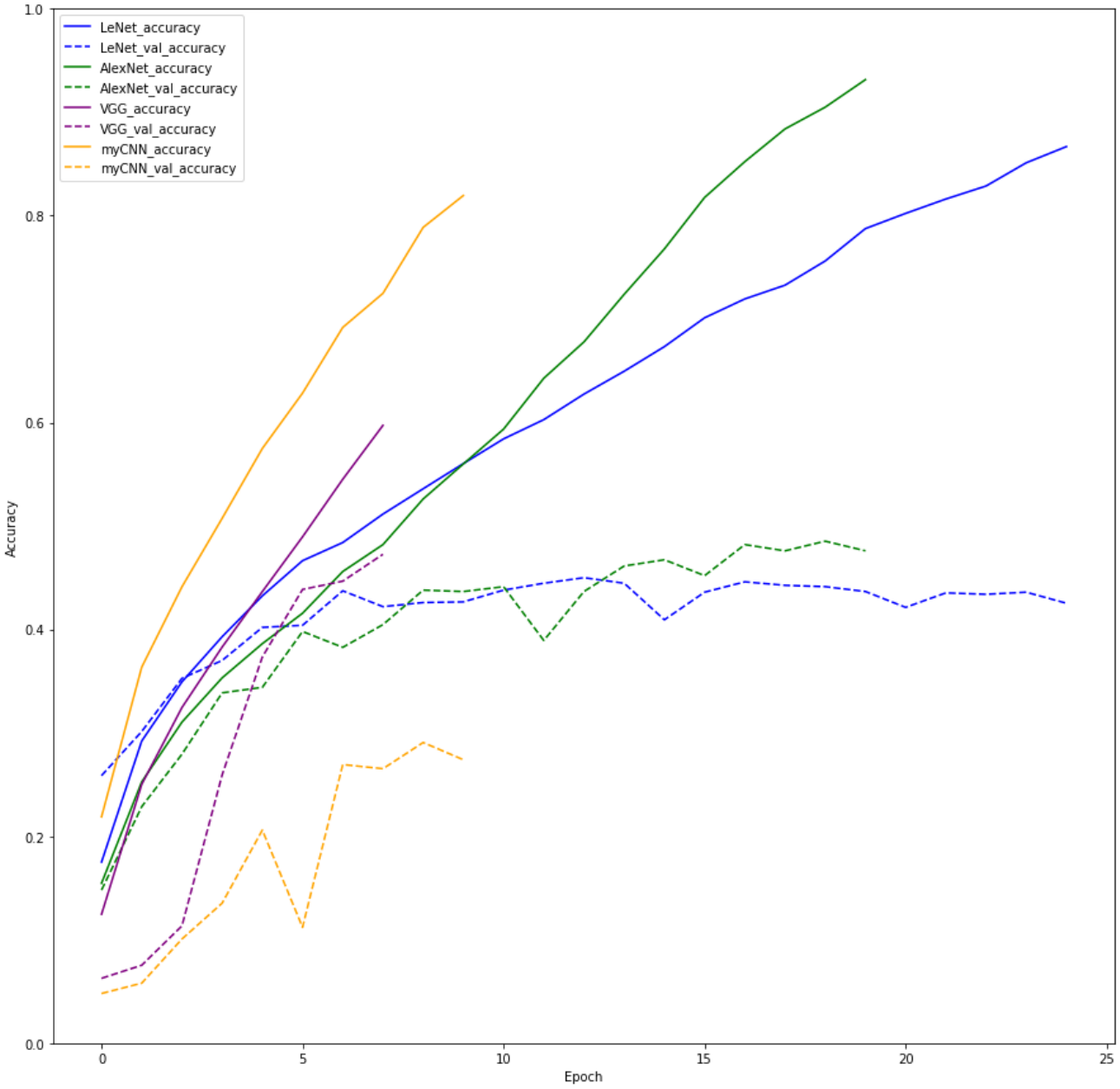
In [ ]:

```python
fig, axs = plt.subplots(2, 2, figsize=(15,15))

axs[0, 0].plot(history.history['accuracy'], label = 'accuracy')
axs[0, 0].plot(history.history['val_accuracy'], label = 'val_accuracy')
axs[0, 0].set_title('LeNet', fontsize=14)
axs[0, 1].plot(alex_history3.history['accuracy'], label = 'accuracy')
axs[0, 1].plot(alex_history3.history['val_accuracy'], label = 'val_accuracy')
axs[0, 1].set_title('AlexNet', fontsize=14)
axs[1, 0].plot(vgghistory.history['accuracy'], label = 'accuracy')
axs[1, 0].plot(vgghistory.history['val_accuracy'], label = 'val_accuracy')
axs[1, 0].set_title('VGG', fontsize=14)
axs[1, 1].plot(my_history2.history['accuracy'], label = 'accuracy')
axs[1, 1].plot(my_history2.history['val_accuracy'], label = 'val_accuracy')
axs[1, 1].set_title('myCNN', fontsize=14)

for ax in axs.flat:
    ax.set(xlabel='Epoch', ylabel='Accuracy')
    ax.set_ylim(0,1)


# Hide x labels and tick labels for top plots and y ticks for right plots.
for ax in axs.flat:
    ax.label_outer()
```



In [ ]:
```
plt.figure(figsize=(15, 15))
plt.plot(history.history['accuracy'], label = 'LeNet_accuracy', color = 'blue')
plt.plot(history.history['val_accuracy'], label = 'LeNet_val_accuracy', color = 'blue', linestyle='dashed')
plt.plot(alex_history3.history['accuracy'], label = 'AlexNet_accuracy', color = 'green')
plt.plot(alex_history3.history['val_accuracy'], label = 'AlexNet_val_accuracy', color = 'green', linestyle='dashed')
plt.plot(vgghistory.history['accuracy'], label = 'VGG_accuracy', color = 'purple')
plt.plot(vgghistory.history['val_accuracy'], label = 'VGG_val_accuracy', color = 'purple', linestyle='dashed')
plt.plot(my_history2.history['accuracy'], label = 'myCNN_accuracy', color = 'orange')
plt.plot(my_history2.history['val_accuracy'], label = 'myCNN_val_accuracy', color = 'orange', linestyle='dashed')


plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='upper left')
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x7f225a3a7610>
```



Αξιολόγηση των αποτελεσμάτων

Επίδραση του πλήθους των δεδομένων/κλάσεων στην απόδοση του μοντέλου

Γενικά, ενα μικρό πλήθος δεδομένων για εκμάθηση του μοντέλου οδηγεί σε χαμηλό accuracy. Ενα μοντέλο με αυστηρούς περιορισμούς σε ενα μικρό training dataset θα έχει σαν αποτέλεσμα να υποεκπαίδευση ενω ενα μοντέλο χωρίς αυστηρούς περιορισμούς θα υπερεκπαιδεύσει το μοντέλο. Σε κάθε περίπτωση θα οδηγηθούμε σε μοντέλα με χαμηλή απόδοση. Πολύ λίγα test data θα έχουν σαν αποτέλεσμα μια αισιόδοξη και υψηλής διακύμανσης εκτίμιση της απόδοσης του μοντέλου. Η εκπαίδευση με πολλά δεδομένα βοηθάει στην αύξηση της ακρίβειας πρόβλεψης του μοντέλου. Τεράστια δεδομένα εκπαίδευσης μπορούν να βοηθήσουν στην αποφυγή του overfitting και γι'αυτό θα κάνουμε χρήση μεθόδων αύξησης των δεδομένων όπως data augmentation που θα δούμε παρακάτω.

Όσον αφορά το πλήθος των κλάσεων, η μείωση του αιρθμού τους εξαρτάται από τον αλγόριθμο και τον τύπο των δεδομένων μας. Συνήθως δεν μπορούμε να εξασφαλίσουμε πως μειώνοντας το πλήθος των κλάσεων θα αυξήσουμε την ακρίβεια πρόβλεψης. Σε πολλές περιπτώσεις ισχύει το αντίθετο, η αύξηση του αριθμού των κλάσεων μπορεί να βελτιώσει την ακρίβεια.

Επίδραση του αλγορίθμου βελτιστοποίησης (optimizer)

Οι Optimizers είναι αλγόριθμοι ή μέθοδοι που χρησιμοποιούνται για να αλλάζουν τα γνωρίσματα των CNN όπως τα βάρη και το learning rate για να μειώσουν το loss. Στα παραπάνω παραδείγματα παρατηρούμε πως η χρήση των Optimizers επηρεάζει σημαντικά τα αποτελέσματά μας. Σε μερικά μοντέλα όπως το LeNet δεν παρατηρήσαμε σημαντική διαφορά αλλά σε άλλα όπως AlexNet και VGG-16 η χρήση διαφορετικών αλγορίθμων Optimizer για τις εικόνες μας έπαιξε καθοριστικό ρόλο κυρίως στην υπερεκπαίδευση αλλά και γενικά στην εκπαίδευση του μοντέλου. Συνεπώς χρειάστηκε πολλές φορές να πειραματιστούμε προτού βρούμε κάποιον αξιόλογο αλγόριθμο. Ο Adam αναφέρεται σε πολλές βιβλιογραφίες σαν ένας από τους καλύτερους αλγορίθμους όμως για τα δικά μας δεδομένα δε μας βοήθησε. Στη δική μας περίπτωση ο adagrad καθώς και ο sgd μας παρείχαν ικανοποιητικά αποτελέσματα.

Επίδραση του μεγέθους δέσμης (batch size)

Η χρήση διαφορετικών μεγεθών από batch sizes σίγουρα επηρεάζει την ακρίβεια όπως διαπιστώσαμε από τα μοντέλα που δοκιμάσαμε. Συνήθως η αύξηση του batch size βοηθάει στην ταχύτερη εκτέλεση του μοντέλους όμως ταυτόχρονα μειώνει την ακρίβεια σε κάποιο ποσοστό. Πιο συγκεκριμένα παρατηρήσαμε πως η χρήση πολύ μεγάλου batch size έχει αρνητική επίδραση στην ακρίβεια του δικτύου κατά τη διάρκεια της εκπαίδευσης καθώς μειώνει τη στοχαστικότητα του gradient descent.

Τις περισσότερες φορές η αύξηση του batch size πράγματι επιτάχυνε την υπολογιστική διαδικασία αλλά δεν αποτελεί πάντα την ιδανικότερη μέθοδο μείωσης του χρόνου εκτέλεσης αφού μπορεί να μειώσει σε σημαντικό βαθμό την ακρίβεια όπως παρατηρήσαμε και από το LeNet μοντέλο. Η χρήση batch sizes των 32, 64 και 128 συνήθως χρειάζεται να δοκιμαστούν προτού επιλέξουμε μοντέλο όμως μερικές φορές η χρήση του 32 ήταν πολύ αργή κυρίως για το VGG το οποίο ευτυχώς παρουσίασε καλύτερα αποτελέσματα για batch size 64 το οποίο και αξιοποιήσαμε.

Να σημειωθεί πως η χρήση μικρότερων batch sizes ο υπολογισμός του σφάλματος περιέχει περισσότερο θόρυβο σε σύγκριση με αυτό που προκύπτει από μεγαλύτερα batch sizes. Αυτός ο θόρυβος μπορεί μερικές φορές να βοηθήσει τον αλγόριθμο να "ξεφύγει" από κάποιο κακό τοπικό ελάχιστο και να βρει κάποιο καλύτερο τοπικό ελάχιστο ή ολικό ελάχιστο.

Αξιολόγηση ορθότητας για test set

Network	Train Acc	Val Acc	Test Acc
Lenet-1	0.8608	0.4433	0.4425
Lenet-2	0.4842	0.3827	0.4040
Lenet-3	0.2874	0.23	0.2430
AlexNet-1	0.8993	0.4560	0.4625
AlexNet-2	0.0460	0.0427	0.0520
AlexNet-3	0.9284	0.4973	0.4885
VGG-1	0.5811	0.4607	0.4635
VGG-2	0.0461	0.0447	0.0505
VGG-3	0.0324	0.0327	0.0360
myCNN-1	0.7789	0.35	0.3580
myCNN-2	0.8222	0.2573	0.2580

Όπως είναι γνωστό το Validation set χρησιμοποιείται για τη ρύθμιση των hyper παραμέτρων, οπότε συνήθως αν αυτή η διαδικασία πραγματοποιείται επιτυχώς θα πρέπει το validation accuracy να είναι καλύτερο από το test accuracy. Στη δική μας υλοποίηση παρατηρούμε πως η ακρίβεια του test set είναι ελαφρώς καλύτερη από αυτή του validation set. Αυτό μπορεί να οφείλεται σε διάφορους παράγοντες. Αρχικά μπορεί να μην πραγματοποιείται σωστά η διαδικασία ρύθμισης των hyperparameters με

αποτέλεσμα να μην βελτιώνεται η ακρίβεια ως προς το validation set. Επίσης μπορεί αυτή η ομοιότητα να οφείλεται σε μικρά σφάλματα. Ακόμα το μικρό test set δεν επαρκεί για αξιόπιστα συμπεράσματα. Πέρα από τα παραπάνω το test set ή γενικότερα οι εικόνες μας μπορεί να αποτελούνται από εύκολα δείγματα σε ενα ποσοστό περίπου 50% με αποτέλεσμα το test set να προσεγγίζει στην ακρίβεια το validation set. Αξίζει όμως να σημειωθεί πως για λόγους χρόνου δεν τρέξαμε τα μοντέλα για αρκετές εποχές ώστε να έχουμε αξιοπρεπή αποτελέσματα, κάτι που αναμφίβολα επηρεάζει τα αποτελέσματά μας.

Εν πάση περιπτώσει, οι διαφορά μεταξύ test set και validation set είναι αμελητέας τάξης και θα μπορούσε να αποδοθεί σε πειραματικό σφάλμα, δεδομένου και του χαμηλού αριθμού epochs.

Να σημειωθεί πως στα παραπάνω δεν μετατρέψαμε την είσοδο στις κατάλληλες διαστάσεις για τα μοντέλα του AlexNet και VGG για τους παρακάτω λόγους:

- **Είχαμε αρκετά προβλήματα με το ipynb το οποίο συνεχώς παρουσίαζε προβλήματα με τη RAM και έκανε επανακκίνηση με αποτέλεσμα να μη μπορούμε να δοκιμάσουμε το μοντέλο όπως πραγματικά θα έπρεπε κυρίως στο VGG.**
- **Στις περιπτώσεις που καταφέραμε να εκτελέσουμε επιτυχώς τα μοντέλα με εισόδους τις κατάλληλες διαστάσεις των εικόνων (μετά από resize), δεν παρατηρήσαμε καλύτερα αποτελέσματα από αυτά που είχαμε για παρόμοιες παραμέτρους χωρίς το resize.**
- **Δεν χρησιμοποιήσαμε τα έτοιμα μοντέλα όπως στο ερώτημα 3 αλλά δημιουργήσαμε το μοντέλο AlexNet και VGG, οπότε τροποποιήσαμε κατάλληλα τα επιπεδά τους για να αντιμετώπισουμε πιθανά προβλήματα, όπως την εξάλειψη κάποιων επιπέδων pooling για να διατηρήσουμε αρκετές παραμέτρους.**
- **Για να επιτύχουμε τα επιθυμητά αποτελέσματα μειώσαμε τα επίπεδα pooling. **

Ερώτημα 2

Βήμα 1: Έλεγχος υπερεκπαίδευσης

Για τον καλύτερο συνδυασμό που λάβατε από το Ερώτημα 1 για μοντέλο σας (MyCNN) και μόνο, δοκιμάστε διάφορους συνδυασμούς των ακόλουθων τεχνικών για τον έλεγχο της υπερεκπαίδευσης (overfitting), όπως:

- **Πρόωρος τερματισμός (early stopping tf.keras.callbacks.EarlyStopping)**
- **Dropout (Dropout)**
- **Επαύξηση δεδομένων (Data augmentation, ImageDataGenerator)**

, ώστε το μοντέλο σας να γενικεύει καλύτερα.

Βήμα 2: Αξιολόγηση

Αξιολογήστε της ορθότητας για το validation και το test set σας.

Επαύξηση δεδομένων

Εκπαιδεύουμε ξανά το αποτελεσματικότερο μοντέλο του MyCNN χρησιμοποιώντας data augmentation. Θα διπλασιάσουμε τον όγκο του dataset:

In [ ]:
```
data_augmentation = tf.keras.Sequential(
  [
    layers.experimental.preprocessing.RandomFlip("horizontal", input_shape=(32, 32, 3)),
    layers.experimental.preprocessing.RandomRotation(0.1),
    layers.experimental.preprocessing.RandomZoom(0.1),
  ]
)
```

In [ ]:
```
print(len(x_train))
```

```
8500
```

In [ ]:
```
augm = data_augmentation(x_train)
print(len(augm))
```

```
8500
```

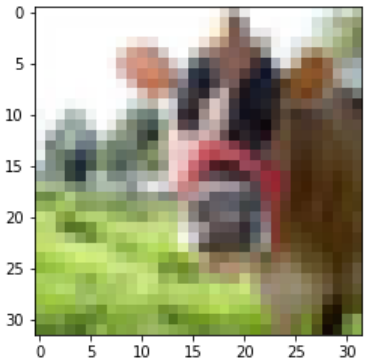
In [ ]:
```
augm_x_train = np.concatenate((x_train, augm))
```

```
print(len(augm_x_train))

augm_y_train = np.concatenate((y_train, np.copy(y_train)))
print(len(augm_y_train))
```

```
17000
17000
```

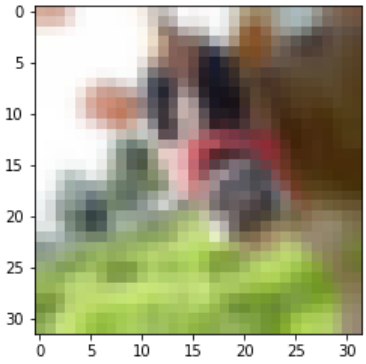
In [ ]:
```
plt.imshow(x_train[0])
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x7fe736d07f50>
```



In [ ]:
```
plt.imshow(augm[0])
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x7fe736e63510>
```



In [ ]:
```
mymodel = tf.keras.models.Sequential()
mymodel.add(DefaultConv2D(64, kernel_size=7, strides=2, input_shape=[224, 224, 3]))
mymodel.add(tf.keras.layers.BatchNormalization())
mymodel.add(tf.keras.layers.Activation("relu"))
mymodel.add(tf.keras.layers.MaxPool2D(pool_size=3, strides=2, padding="SAME"))
prev_filters = 64
for filters in [64] * 3 + [128] * 4 + [256] * 6 + [512] * 3:
    strides = 1 if filters == prev_filters else 2
    mymodel.add(ResidualUnit(filters, strides=strides))
    prev_filters = filters
mymodel.add(tf.keras.layers.GlobalAvgPool2D())
mymodel.add(tf.keras.layers.Flatten())
mymodel.add(tf.keras.layers.Dense(20, activation="softmax"))
```

In [ ]:
```
mymodel.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

augm_history = mymodel.fit(augm_x_train, augm_y_train, epochs=10, batch_size=128,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/10
WARNING:tensorflow:Model was constructed with shape (None, 224, 224, 3) for input KerasTensor(type_spec=TensorSpec(shape
=(None, 224, 224, 3), dtype=tf.float32, name='conv2d_72_input'), name='conv2d_72_input', description="created by layer
'conv2d_72_input'"), but it was called on an input with incompatible shape (None, 32, 32, 3).
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/backend.py:4930: UserWarning: "`sparse_categorical_crosse
ntropy` received `from_logits=True`, but the `output` argument was produced by a sigmoid or softmax activation and thus
does not represent logits. Was this intended?"
  '"`sparse_categorical_crossentropy` received `from_logits=True`, but '
```

```
WARNING:tensorflow:Model was constructed with shape (None, 224, 224, 3) for input KerasTensor(type_spec=TensorSpec(shape
=(None, 224, 224, 3), dtype=tf.float32, name='conv2d_72_input'), name='conv2d_72_input', description="created by layer
'conv2d_72_input'"), but it was called on an input with incompatible shape (None, 32, 32, 3).
133/133 [==============================] - ETA: 0s - loss: 2.5453 - accuracy: 0.2681WARNING:tensorflow:Model was constru
cted with shape (None, 224, 224, 3) for input KerasTensor(type_spec=TensorSpec(shape=(None, 224, 224, 3), dtype=tf.float
32, name='conv2d_72_input'), name='conv2d_72_input', description="created by layer 'conv2d_72_input'"), but it was calle
d on an input with incompatible shape (None, 32, 32, 3).
133/133 [==============================] - 1023s 8s/step - loss: 2.5453 - accuracy: 0.2681 - val_loss: 3.7088 - val_accu
racy: 0.0727
Epoch 2/10
133/133 [==============================] - 1008s 8s/step - loss: 1.9603 - accuracy: 0.4044 - val_loss: 3.0861 - val_accu
racy: 0.1333
Epoch 3/10
133/133 [==============================] - 1010s 8s/step - loss: 1.6570 - accuracy: 0.4849 - val_loss: 3.2579 - val_accu
racy: 0.2040
Epoch 4/10
133/133 [==============================] - 1008s 8s/step - loss: 1.4424 - accuracy: 0.5464 - val_loss: 2.5803 - val_accu
racy: 0.3373
Epoch 5/10
133/133 [==============================] - 1006s 8s/step - loss: 1.2348 - accuracy: 0.6098 - val_loss: 2.2532 - val_accu
racy: 0.3933
Epoch 6/10
133/133 [==============================] - 1008s 8s/step - loss: 1.0351 - accuracy: 0.6684 - val_loss: 2.2964 - val_accu
racy: 0.4027
Epoch 7/10
133/133 [==============================] - 1006s 8s/step - loss: 0.8916 - accuracy: 0.7114 - val_loss: 2.8124 - val_accu
racy: 0.3347
Epoch 8/10
133/133 [==============================] - 1006s 8s/step - loss: 0.7519 - accuracy: 0.7552 - val_loss: 2.6258 - val_accu
racy: 0.3500
Epoch 9/10
133/133 [==============================] - 1002s 8s/step - loss: 0.5801 - accuracy: 0.8096 - val_loss: 3.0492 - val_accu
racy: 0.3567
Epoch 10/10
133/133 [==============================] - 1001s 8s/step - loss: 0.4639 - accuracy: 0.8459 - val_loss: 2.6441 - val_accu
racy: 0.4413
```

Έλεγχος με το test set

In [ ]:
```
test_loss, test_acc = mymodel.evaluate(x_test,  y_test, verbose = 1)
```

```
63/63 [==============================] - 7s 107ms/step - loss: 2.6284 - accuracy: 0.4420
```

Dropout

Προσθέτουμε ένα επίπεδο dropout με συντελεστή 20%, μετά το pooling. Αυτό σημαίνει ότι τα βάρη του 20% των νευρώνων θα αγνοηθούν από το επόμενο επίπεδο, με τυχαία επιλογή, ώστε να μην αποκτούν συγκεκριμένα βάρη υπερβολικά μεγάλη επιρροή στο αποτέλεσμα.

In [ ]:
```
mymodel = tf.keras.models.Sequential()
mymodel.add(DefaultConv2D(64, kernel_size=7, strides=2, input_shape=[224, 224, 3]))
mymodel.add(tf.keras.layers.BatchNormalization())
mymodel.add(tf.keras.layers.Activation("relu"))
mymodel.add(tf.keras.layers.MaxPool2D(pool_size=3, strides=2, padding="SAME"))
prev_filters = 64
for filters in [64] * 3 + [128] * 4 + [256] * 6 + [512] * 3:
    strides = 1 if filters == prev_filters else 2
    mymodel.add(ResidualUnit(filters, strides=strides))
    prev_filters = filters
mymodel.add(tf.keras.layers.GlobalAvgPool2D())
##
mymodel.add(tf.keras.layers.Dropout(0.2))
##
mymodel.add(tf.keras.layers.Flatten())
mymodel.add(tf.keras.layers.Dense(20, activation="softmax"))
```

In [ ]:
```
mymodel.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

my_history = mymodel.fit(x_train, y_train, epochs=10, batch_size=128, validation_data=(x_val, y_val))
```

```
Epoch 1/10
WARNING:tensorflow:Model was constructed with shape (None, 224, 224, 3) for input KerasTensor(type_spec=TensorSpec(shape
=(None, 224, 224, 3), dtype=tf.float32, name='conv2d_72_input'), name='conv2d_72_input', description="created by layer
'conv2d_72_input'"), but it was called on an input with incompatible shape (None, 32, 32, 3).
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/backend.py:4930: UserWarning: "`sparse_categorical_crosse
ntropy` received `from_logits=True`, but the `output` argument was produced by a sigmoid or softmax activation and thus
does not represent logits. Was this intended?"
  '"`sparse_categorical_crossentropy` received `from_logits=True`, but '
WARNING:tensorflow:Model was constructed with shape (None, 224, 224, 3) for input KerasTensor(type_spec=TensorSpec(shape
=(None, 224, 224, 3), dtype=tf.float32, name='conv2d_72_input'), name='conv2d_72_input', description="created by layer
'conv2d_72_input'"), but it was called on an input with incompatible shape (None, 32, 32, 3).
67/67 [==============================] - ETA: 0s - loss: 3.0708 - accuracy: 0.1947WARNING:tensorflow:Model was construct
ed with shape (None, 224, 224, 3) for input KerasTensor(type_spec=TensorSpec(shape=(None, 224, 224, 3), dtype=tf.float3
2, name='conv2d_72_input'), name='conv2d_72_input', description="created by layer 'conv2d_72_input'"), but it was called
on an input with incompatible shape (None, 32, 32, 3).
67/67 [==============================] - 462s 7s/step - loss: 3.0708 - accuracy: 0.1947 - val_loss: 5.5089 - val_accurac
y: 0.0487
Epoch 2/10
67/67 [==============================] - 450s 7s/step - loss: 2.3634 - accuracy: 0.3371 - val_loss: 3.5503 - val_accurac
y: 0.0693
Epoch 3/10
67/67 [==============================] - 451s 7s/step - loss: 2.0079 - accuracy: 0.4052 - val_loss: 3.9938 - val_accurac
y: 0.0820
Epoch 4/10
67/67 [==============================] - 446s 7s/step - loss: 1.9228 - accuracy: 0.4516 - val_loss: 3.3978 - val_accurac
y: 0.1267
Epoch 5/10
67/67 [==============================] - 444s 7s/step - loss: 1.7541 - accuracy: 0.4949 - val_loss: 3.0414 - val_accurac
y: 0.1787
Epoch 6/10
67/67 [==============================] - 455s 7s/step - loss: 1.4283 - accuracy: 0.5667 - val_loss: 3.7755 - val_accurac
y: 0.1573
Epoch 7/10
67/67 [==============================] - 452s 7s/step - loss: 1.1670 - accuracy: 0.6329 - val_loss: 2.8873 - val_accurac
y: 0.2967
Epoch 8/10
67/67 [==============================] - 447s 7s/step - loss: 1.2128 - accuracy: 0.6258 - val_loss: 5.6351 - val_accurac
y: 0.2193
Epoch 9/10
67/67 [==============================] - 459s 7s/step - loss: 1.0967 - accuracy: 0.6624 - val_loss: 2.5791 - val_accurac
y: 0.3393
Epoch 10/10
67/67 [==============================] - 451s 7s/step - loss: 0.8192 - accuracy: 0.7448 - val_loss: 2.5227 - val_accurac
y: 0.3987
```

Έλεγχος με το test set

In [ ]:
```
test_loss, test_acc = mymodel.evaluate(x_test,  y_test, verbose = 1)
```

```
63/63 [==============================] - 7s 109ms/step - loss: 2.0205 - accuracy: 0.4270
```

Early stopping

Διακόπτουμε την εκπαίδευση το δικτύου εάν δεν παρατηρηθεί αλλαγή στην απώλεια για τρεις διαδοχικές epochs.

In [ ]:
```
mymodel = tf.keras.models.Sequential()
mymodel.add(DefaultConv2D(64, kernel_size=7, strides=2, input_shape=[224, 224, 3]))
mymodel.add(tf.keras.layers.BatchNormalization())
mymodel.add(tf.keras.layers.Activation("relu"))
mymodel.add(tf.keras.layers.MaxPool2D(pool_size=3, strides=2, padding="SAME"))
prev_filters = 64
for filters in [64] * 3 + [128] * 4 + [256] * 6 + [512] * 3:
    strides = 1 if filters == prev_filters else 2
    mymodel.add(ResidualUnit(filters, strides=strides))
    prev_filters = filters
mymodel.add(tf.keras.layers.GlobalAvgPool2D())
mymodel.add(tf.keras.layers.Flatten())
mymodel.add(tf.keras.layers.Dense(20, activation="softmax"))
```

In [ ]:
```
callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)
# This callback will stop the training when there is no improvement in
# the loss for three consecutive epochs.
mymodel.compile(optimizer='adam', loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits = False), metrics = ['a

my_history = mymodel.fit(x_train, y_train, epochs=10, batch_size=128, callbacks=[callback], validation_data=(x_val, y_va
```

```
Epoch 1/10
WARNING:tensorflow:Model was constructed with shape (None, 224, 224, 3) for input KerasTensor(type_spec=TensorSpec(shape
=(None, 224, 224, 3), dtype=tf.float32, name='conv2d_108_input'), name='conv2d_108_input', description="created by layer
'conv2d_108_input'"), but it was called on an input with incompatible shape (None, 32, 32, 3).
WARNING:tensorflow:Model was constructed with shape (None, 224, 224, 3) for input KerasTensor(type_spec=TensorSpec(shape
=(None, 224, 224, 3), dtype=tf.float32, name='conv2d_108_input'), name='conv2d_108_input', description="created by layer
'conv2d_108_input'"), but it was called on an input with incompatible shape (None, 32, 32, 3).
67/67 [==============================] - ETA: 0s - loss: 2.7435 - accuracy: 0.2308WARNING:tensorflow:Model was construct
ed with shape (None, 224, 224, 3) for input KerasTensor(type_spec=TensorSpec(shape=(None, 224, 224, 3), dtype=tf.float3
2, name='conv2d_108_input'), name='conv2d_108_input', description="created by layer 'conv2d_108_input'"), but it was cal
led on an input with incompatible shape (None, 32, 32, 3).
67/67 [==============================] - 462s 7s/step - loss: 2.7435 - accuracy: 0.2308 - val_loss: 3.7716 - val_accurac
y: 0.0387
Epoch 2/10
67/67 [==============================] - 451s 7s/step - loss: 2.0930 - accuracy: 0.3698 - val_loss: 3.2658 - val_accurac
y: 0.0833
Epoch 3/10
67/67 [==============================] - 453s 7s/step - loss: 1.8345 - accuracy: 0.4400 - val_loss: 3.8931 - val_accurac
y: 0.0507
Epoch 4/10
67/67 [==============================] - 459s 7s/step - loss: 1.5817 - accuracy: 0.5153 - val_loss: 3.9834 - val_accurac
y: 0.0700
Epoch 5/10
67/67 [==============================] - 458s 7s/step - loss: 1.3917 - accuracy: 0.5667 - val_loss: 3.4393 - val_accurac
y: 0.1367
Epoch 6/10
67/67 [==============================] - 452s 7s/step - loss: 1.1438 - accuracy: 0.6381 - val_loss: 3.1502 - val_accurac
y: 0.2347
Epoch 7/10
67/67 [==============================] - 462s 7s/step - loss: 0.9754 - accuracy: 0.6875 - val_loss: 2.9818 - val_accurac
y: 0.2627
Epoch 8/10
67/67 [==============================] - 454s 7s/step - loss: 0.7961 - accuracy: 0.7432 - val_loss: 3.6183 - val_accurac
y: 0.2347
Epoch 9/10
67/67 [==============================] - 460s 7s/step - loss: 0.6641 - accuracy: 0.7858 - val_loss: 2.9082 - val_accurac
y: 0.3220
Epoch 10/10
67/67 [==============================] - 454s 7s/step - loss: 0.5580 - accuracy: 0.8171 - val_loss: 3.1420 - val_accurac
y: 0.3447
```

Έλεγχος με το test set

In [ ]:
```
test_loss, test_acc = mymodel.evaluate(x_test,  y_test, verbose = 1)
```

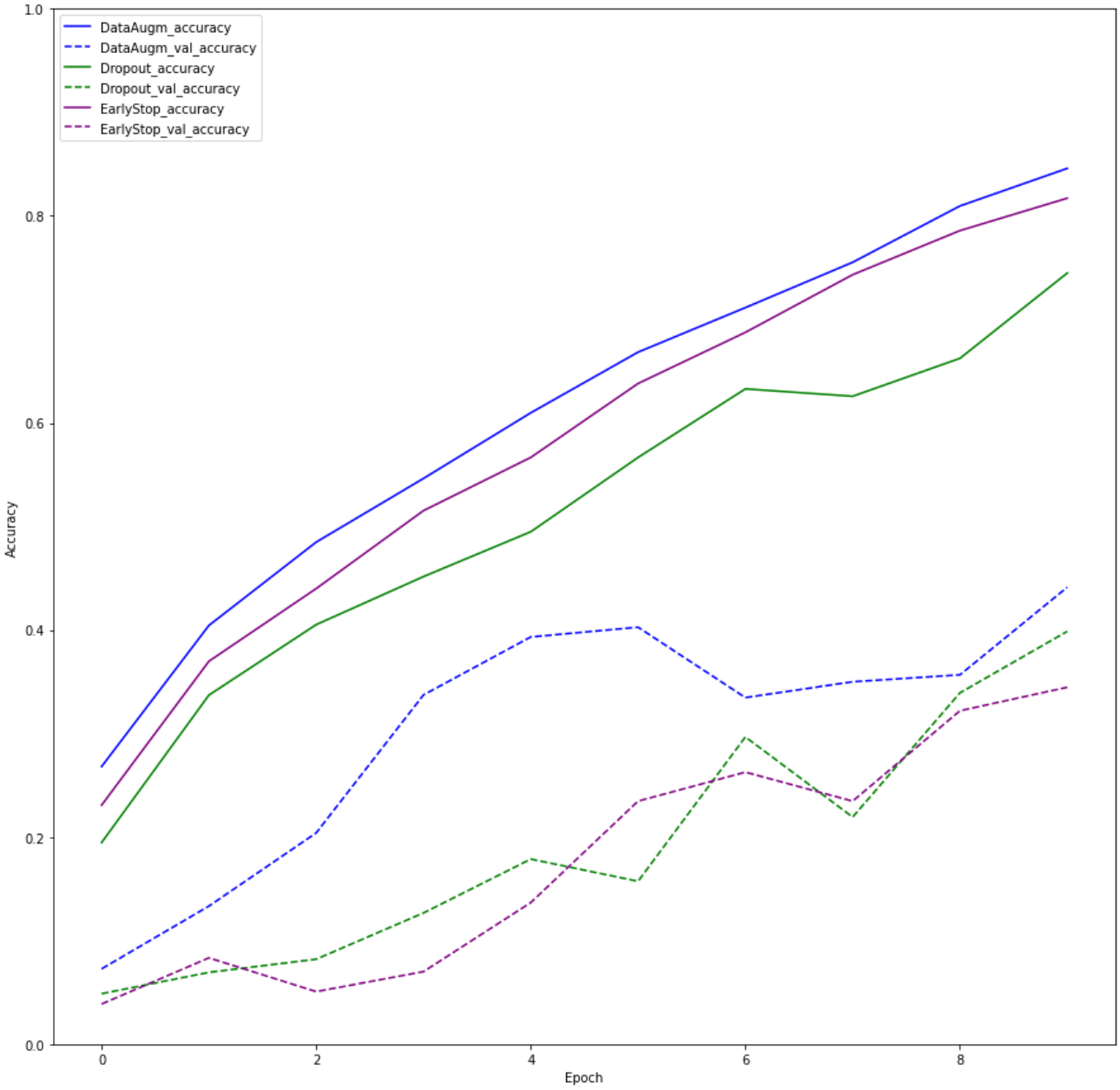
```
63/63 [==============================] - 7s 107ms/step - loss: 2.9081 - accuracy: 0.3330
```

Παρατηρούμε πως οι μέθοδοι αυτές μειώνουν την ακρίβεια (accuracy) του μοντέλου, όμως παράλληλα περιορίζουν το φαινόμενο της υπερεκπαίδευσης, καθώς η τιμή της validation_accuracy, η οποία προκύπτει από διαφορετικό dataset επικύρωσης, είναι πλησιέστερη στο accuracy από ότι στο αρχικό μοντέλο. Εκ των αποτελεσμάτων, αποδοτικότερη φαίνεται πως είναι η μέθοδος του dropout (ως προς την ελάττωση του overfitting) ή της επαύξησης δεδομένων, ως προς την ακρίβεια.

Method	Train Acc	Val Acc	Test Acc
Data augmentation	0.8459	0.4413	0.4420
Dropout	0.7448	0.3987	0.4270
Early stopping	0.8171	0.3447	0.3330

In [ ]:
```
plt.figure(figsize=(15, 15))
plt.plot(augm_history.history['accuracy'], label = 'DataAugm_accuracy', color = 'blue')
plt.plot(augm_history.history['val_accuracy'], label = 'DataAugm_val_accuracy', color = 'blue', linestyle='dashed')
plt.plot(drop_history.history['accuracy'], label = 'Dropout_accuracy', color = 'green')
plt.plot(drop_history.history['val_accuracy'], label = 'Dropout_val_accuracy', color = 'green', linestyle='dashed')
plt.plot(es_history.history['accuracy'], label = 'EarlyStop_accuracy', color = 'purple')
plt.plot(es_history.history['val_accuracy'], label = 'EarlyStop_val_accuracy', color = 'purple', linestyle='dashed')

plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='upper left')
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x7fb3f325eb90>
```



Ερώτημα 3

Βήμα 1: Μεταφορά γνώσης

Εφαρμόστε μεταφορά γνώσης (transfer learning) στο δικό σας μοντέλο (MyCNN), που αξιολογήσατε ως καλύτερο προς την ακρίβεια στην αντιμετώπιση της υπερεκπαίδεσης.

Για το transfer learning επιλέξτε το VGG16 για μεταφορά μάθησης.

1. **"Παγώστε" τη συνελικτική βάση και εκπαιδεύστε την κεφαλή ταξινόμησης (classification head - σημαία trainable = False).**
2. **Εκπαιδέστε μόνο ένα ποσοστό των επιπέδων, το οποίο βρίσκεται προς την έξοδο του δικτύου. Οι σημαίες trainable εδώ θα πρέπει να οριστούν ανά επίπεδο.**
3. **Αξιολογήστε τα αποτελέσματά σας, βάσει της ορθότητας του validation set και του test set σας.**

Θα χρησιμοποιήσουμε dropout για την αντιμετώπιση της υπερεκπαίδευσης.

1η Μέθοδος

Παγώστε τη συνελικτική βάση και εκπαιδεύστε την κεφαλή ταξινόμησης

Δημιουργία headless μοντέλου (συνελικτική βάση VGG16)

In [ ]:
```
from tensorflow.keras.applications import vgg16


#feature_extractor_layer.trainable = False
# Init the VGG model
vgg_conv = vgg16.VGG16(weights='imagenet', include_top=False, input_shape=(32,32,3))

# Freeze all the layers
for layer in vgg_conv.layers[:]:
    layer.trainable = False

# Check the trainable status of the individual layers
for layer in vgg_conv.layers:
    print(layer, layer.trainable)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_t
f_kernels_notop.h5
58892288/58889256 [==============================] - 1s 0us/step
<tensorflow.python.keras.engine.input_layer.InputLayer object at 0x7f9e5f356990> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f9e5ee01950> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f9e5edefc90> False
<tensorflow.python.keras.layers.pooling.MaxPooling2D object at 0x7f9e5ed9a650> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f9e5f5b4790> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f9e5a595910> False
<tensorflow.python.keras.layers.pooling.MaxPooling2D object at 0x7f9e5ed99f10> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f9e5a59bb50> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f9e5a5a20d0> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f9e5a5a6ed0> False
<tensorflow.python.keras.layers.pooling.MaxPooling2D object at 0x7f9e5a59bcd0> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f9e5a5ad210> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f9e5ee14450> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f9e5a5a0310> False
<tensorflow.python.keras.layers.pooling.MaxPooling2D object at 0x7f9e5a595710> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f9e5a536650> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f9e5ee43990> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f9e5a5925d0> False
<tensorflow.python.keras.layers.pooling.MaxPooling2D object at 0x7f9e5a540c10> False
```

Επισύναψη του MyCNN

In [ ]:
```
from functools import partial
```

In [ ]:
```
## MY CNN
DefaultConv2D = partial(tf.keras.layers.Conv2D, kernel_size=3, strides=1, padding="SAME", use_bias=False)
class ResidualUnit(tf.keras.layers.Layer):
    def __init__(self, filters, strides=1, activation="relu", **kwargs):
        super().__init__(**kwargs)
        self.activation = tf.keras.activations.get(activation)
        self.main_layers = [DefaultConv2D(filters, strides=strides), tf.keras.layers.BatchNormalization(), self.activati
        self.skip_layers = []
        if strides > 1:
            self.skip_layers = [DefaultConv2D(filters, kernel_size=1, strides=strides), tf.keras.layers.BatchNormalizati

    def call(self, inputs):
        Z = inputs
        for layer in self.main_layers:
            Z = layer(Z)
        skip_Z = inputs
        for layer in self.skip_layers:
            skip_Z = layer(skip_Z)
        return self.activation(Z + skip_Z)
```

In [ ]:
```
mymodel = tf.keras.models.Sequential()
mymodel.add(DefaultConv2D(64, kernel_size=7, strides=2, input_shape=[7, 7, 512]))
mymodel.add(tf.keras.layers.BatchNormalization())
mymodel.add(tf.keras.layers.Activation("relu"))
# mymodel.add(tf.keras.layers.MaxPool2D(pool_size=3, strides=2, padding="SAME"))
prev_filters = 64
for filters in [64] * 3 + [128] * 4 + [256] * 6 + [512] * 3:
    strides = 1 if filters == prev_filters else 2
    mymodel.add(ResidualUnit(filters, strides=strides))
```

```
    prev_filters = filters
# mymodel.add(tf.keras.layers.GlobalAvgPool2D())
##
mymodel.add(tf.keras.layers.Dropout(0.2))
##
mymodel.add(tf.keras.layers.Flatten())
mymodel.add(tf.keras.layers.Dense(20, activation="softmax"))
```

In [ ]:
```
tmodel = tf.keras.Sequential([
  vgg_conv,
  mymodel
])

tmodel.summary()
```

```
WARNING:tensorflow:Model was constructed with shape (None, 7, 7, 512) for input KerasTensor(type_spec=TensorSpec(shape=
(None, 7, 7, 512), dtype=tf.float32, name='conv2d_input'), name='conv2d_input', description="created by layer 'conv2d_in
put'"), but it was called on an input with incompatible shape (None, 1, 1, 512).
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Functional)           (None, 1, 1, 512)         14714688
_________________________________________________________________
sequential (Sequential)      (None, 20)                22908180
=================================================================
Total params: 37,622,868
Trainable params: 22,891,156
Non-trainable params: 14,731,712
_________________________________________________________________
```

Εκπαίδευση με τις βέλτιστες παραμέτρους που προέκυψαν από τις προηγούμενες εκδόσεις του MyCNN

In [ ]:
```
tmodel.compile(optimizer='adam',
               loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
               metrics=['accuracy'])

thistory = tmodel.fit(x_train, y_train, epochs=8, batch_size=128, validation_data=(x_val, y_val))
```

```
Epoch 1/8
WARNING:tensorflow:Model was constructed with shape (None, 7, 7, 512) for input KerasTensor(type_spec=TensorSpec(shape=
(None, 7, 7, 512), dtype=tf.float32, name='conv2d_input'), name='conv2d_input', description="created by layer 'conv2d_in
put'"), but it was called on an input with incompatible shape (None, 1, 1, 512).
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/backend.py:4930: UserWarning: "`sparse_categorical_crosse
ntropy` received `from_logits=True`, but the `output` argument was produced by a sigmoid or softmax activation and thus
does not represent logits. Was this intended?"
  '"`sparse_categorical_crossentropy` received `from_logits=True`, but '
WARNING:tensorflow:Model was constructed with shape (None, 7, 7, 512) for input KerasTensor(type_spec=TensorSpec(shape=
(None, 7, 7, 512), dtype=tf.float32, name='conv2d_input'), name='conv2d_input', description="created by layer 'conv2d_in
put'"), but it was called on an input with incompatible shape (None, 1, 1, 512).
67/67 [==============================] - ETA: 0s - loss: 2.9983 - accuracy: 0.1769WARNING:tensorflow:Model was construct
ed with shape (None, 7, 7, 512) for input KerasTensor(type_spec=TensorSpec(shape=(None, 7, 7, 512), dtype=tf.float32, na
me='conv2d_input'), name='conv2d_input', description="created by layer 'conv2d_input'"), but it was called on an input w
ith incompatible shape (None, 1, 1, 512).
67/67 [==============================] - 496s 7s/step - loss: 2.9983 - accuracy: 0.1769 - val_loss: 3.0252 - val_accurac
y: 0.0407
Epoch 2/8
67/67 [==============================] - 492s 7s/step - loss: 2.2822 - accuracy: 0.3004 - val_loss: 3.0565 - val_accurac
y: 0.0407
Epoch 3/8
67/67 [==============================] - 492s 7s/step - loss: 2.0846 - accuracy: 0.3698 - val_loss: 2.9995 - val_accurac
y: 0.0540
Epoch 4/8
67/67 [==============================] - 492s 7s/step - loss: 1.9729 - accuracy: 0.4039 - val_loss: 2.7724 - val_accurac
y: 0.1333
Epoch 5/8
67/67 [==============================] - 492s 7s/step - loss: 1.8176 - accuracy: 0.4556 - val_loss: 2.4296 - val_accurac
y: 0.2273
Epoch 6/8
67/67 [==============================] - 492s 7s/step - loss: 1.8271 - accuracy: 0.4516 - val_loss: 2.0404 - val_accurac
y: 0.3747
Epoch 7/8
67/67 [==============================] - 494s 7s/step - loss: 1.6070 - accuracy: 0.5078 - val_loss: 1.9159 - val_accurac
y: 0.4207
Epoch 8/8
67/67 [==============================] - 491s 7s/step - loss: 1.5491 - accuracy: 0.5274 - val_loss: 1.9432 - val_accurac
y: 0.4207
```

In [ ]:
```
tLoss, tAcc = tmodel.evaluate(x_test,  y_test, verbose=2)
```

```
63/63 - 24s - loss: 1.8568 - accuracy: 0.4365
```

2η Μέθοδος

Εκπαιδεύστε μόνο ένα ποσοστό των επιπέδων, το οποίο βρίσκεται προς την έξοδο του δικτύου. Οι σημαίες trainable εδώ θα πρέπει να οριστούν ανά επίπεδο.

Για την εφαρμογή του Fine Tuning είναι σημαντικό να το εκπαιδεύσουμε αφού το μοντέλο με τα frozen layers έχει συγκλίνει. Επίσης είναι σημαντικό να χρησιμοποιήσουμε πολύ μικρό learning rate για το μοντέλο μας γιατί εκπαιδεύουμε ενα πολύ μεγαλύτερο μοντέλο από το προηγούμενο το οποίο είναι ευαίσθητο σε overfitting.

In [ ]:
```
# Freeze all the layers
for layer in vgg_conv.layers[:]:
    layer.trainable = False

# unfreeze 4 top layers (before the head)
vgg_conv.layers[11].trainable = True
vgg_conv.layers[12].trainable = True
vgg_conv.layers[13].trainable = True
vgg_conv.layers[14].trainable = True
vgg_conv.layers[15].trainable = True
vgg_conv.layers[16].trainable = True
vgg_conv.layers[17].trainable = True
vgg_conv.layers[18].trainable = True

# Check the trainable status of the individual layers
for layer in vgg_conv.layers:
    print(layer, layer.trainable)

  # Fine tuning
# Unfreeze the base model
#feature_extractor_layer.trainable = True
# It's important to recompile your model after you make any changes
# to the `trainable` attribute of any inner layer, so that your changes
# are take into account



tmodel.compile(optimizer=tf.keras.optimizers.Adam(1e-5),  # Very low learning rate
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

# Train end-to-end. Be careful to stop before you overfit!
thistoryFineTuning = tmodel.fit(x_train, y_train, epochs=8, batch_size=128, validation_data=(x_val, y_val))
```

```
<tensorflow.python.keras.engine.input_layer.InputLayer object at 0x7f01d22cca90> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f01d1e113d0> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f01d1e1c310> False
<tensorflow.python.keras.layers.pooling.MaxPooling2D object at 0x7f01d0572750> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f01d0576d90> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f01ccd9d510> False
<tensorflow.python.keras.layers.pooling.MaxPooling2D object at 0x7f01d05728d0> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f01ccda4750> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f01ccdaa390> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f01ccda4bd0> False
<tensorflow.python.keras.layers.pooling.MaxPooling2D object at 0x7f01d23941d0> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f01ccdb5fd0> True
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f01ccdb9f10> True
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f01ccda1bd0> True
<tensorflow.python.keras.layers.pooling.MaxPooling2D object at 0x7f01ccd99b50> True
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f01ccdbec90> True
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f01ccd9d9d0> True
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f01ccdc31d0> True
<tensorflow.python.keras.layers.pooling.MaxPooling2D object at 0x7f01ccdcc210> True
Epoch 1/8
WARNING:tensorflow:Model was constructed with shape (None, 7, 7, 512) for input KerasTensor(type_spec=TensorSpec(shape=
(None, 7, 7, 512), dtype=tf.float32, name='conv2d_input'), name='conv2d_input', description="created by layer 'conv2d_in
put'"), but it was called on an input with incompatible shape (None, 1, 1, 512).
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/backend.py:4930: UserWarning: "`sparse_categorical_crosse
ntropy` received `from_logits=True`, but the `output` argument was produced by a sigmoid or softmax activation and thus
does not represent logits. Was this intended?"
  '"`sparse_categorical_crossentropy` received `from_logits=True`, but '
```

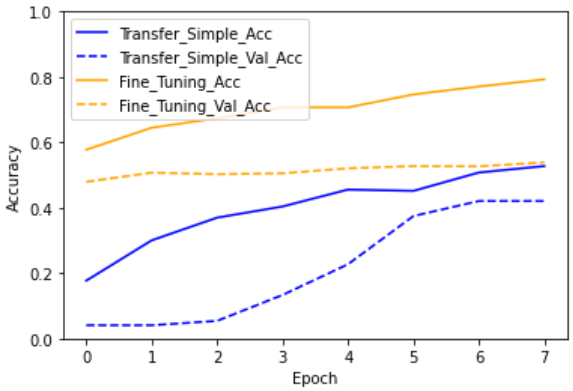
```
WARNING:tensorflow:Model was constructed with shape (None, 7, 7, 512) for input KerasTensor(type_spec=TensorSpec(shape=
(None, 7, 7, 512), dtype=tf.float32, name='conv2d_input'), name='conv2d_input', description="created by layer 'conv2d_in
put'"), but it was called on an input with incompatible shape (None, 1, 1, 512).
67/67 [==============================] - ETA: 0s - loss: 1.4024 - accuracy: 0.5775 WARNING:tensorflow:Model was construc
ted with shape (None, 7, 7, 512) for input KerasTensor(type_spec=TensorSpec(shape=(None, 7, 7, 512), dtype=tf.float32, n
ame='conv2d_input'), name='conv2d_input', description="created by layer 'conv2d_input'"), but it was called on an input
with incompatible shape (None, 1, 1, 512).
67/67 [==============================] - 810s 12s/step - loss: 1.4024 - accuracy: 0.5775 - val_loss: 1.7098 - val_accura
cy: 0.4793
Epoch 2/8
67/67 [==============================] - 813s 12s/step - loss: 1.1688 - accuracy: 0.6447 - val_loss: 1.6428 - val_accura
cy: 0.5073
Epoch 3/8
67/67 [==============================] - 816s 12s/step - loss: 1.0488 - accuracy: 0.6735 - val_loss: 1.6378 - val_accura
cy: 0.5027
Epoch 4/8
67/67 [==============================] - 810s 12s/step - loss: 0.9736 - accuracy: 0.7069 - val_loss: 1.6190 - val_accura
cy: 0.5053
Epoch 5/8
67/67 [==============================] - 804s 12s/step - loss: 0.9338 - accuracy: 0.7068 - val_loss: 1.6179 - val_accura
cy: 0.5207
Epoch 6/8
67/67 [==============================] - 806s 12s/step - loss: 0.8141 - accuracy: 0.7465 - val_loss: 1.6016 - val_accura
cy: 0.5273
Epoch 7/8
67/67 [==============================] - 806s 12s/step - loss: 0.7432 - accuracy: 0.7708 - val_loss: 1.6462 - val_accura
cy: 0.5267
Epoch 8/8
67/67 [==============================] - 811s 12s/step - loss: 0.6679 - accuracy: 0.7926 - val_loss: 1.6739 - val_accura
cy: 0.5387
```

In [ ]:
```
tLossFineTuning, tAccFineTuning = tmodel.evaluate(x_test,  y_test, verbose=2)
```

```
63/63 - 17s - loss: 1.6044 - accuracy: 0.5460
```

In [ ]:
```
plt.plot(thistory.history['accuracy'], label='Transfer_Simple_Acc', color = 'blue')
plt.plot(thistory.history['val_accuracy'], label = 'Transfer_Simple_Val_Acc', color = 'blue', linestyle='dashed')
plt.plot(thistoryFineTuning.history['accuracy'], label='Fine_Tuning_Acc', color = 'orange')
plt.plot(thistoryFineTuning.history['val_accuracy'], label = 'Fine_Tuning_Val_Acc', color = 'orange', linestyle='dashed'

plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='upper left')
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x7f01c83d8c90>
```



Αξιολόγηση

Παρατηρούμε πως εφαρμόζοντας fine tuning σε ορισμένα επίπεδα του δικτύου επιτυγχάνεται μεγαλύτερη ακρίβεια, το οποίο είναι αναμενόμενο, καθώς εκπαιδεύουμε περισσότερα layers με το δικό μας dataset. Η ακρίβεια που προκύπτει από το transfer learning παραμένει, ωστόσο, χαμηλότερη αυτής που προκύπτει από την εκπαίδευση ενός ανάλογου δικτύου απευθείας πάνω στο dataset, όπως είδαμε στο ερώτημα 1. Αυτό θα μπορούσε να οφείλεται στο γεγονός πως, παρ' όλο που η συνελικτική βάση έχει εκπαιδευθεί σε αρκετά παρόμοιο με το cifar100 dataset (imagenet), οι εικόνες αυτού είναι σημαντικά υψηλότερης ανάλυσης.

Method	Train Acc	Val Acc	Test Acc
Freezing convolutional base	0.5274	0.4207	0.4365

Method	Train Acc	Val Acc	Test Acc
Fine tuning	0.7926	0.5387	0.5460

Διαχείριση μνήμης (TFRecord)

Η φόρτωση δεδομένων με τον τρόπο που το κάναμε παραπάνω στο απλό παράδειγμα υλοποίησης είναι πολύ βολική αλλά δεν είναι αποτελεσματική ως προς τη διαχείριση της μνήμης. Συγκεκριμένα, με τον τρόπο αυτό, τα δεδομένα αποθηκεύονται απευθείας σε μεταβλητές, οι οποίες όλες μαζί καταλαμβάνουν τη RAM της CPU ή της GPU, κάτι που κάνει αδύνατη τη διαχείριση μεγάλων datasets ή τον μεταχηματισμό των δεδομένων όπως όταν κάνουμε αύξηση δεδομένων (data augmentation).

Για να παρακαμφθεί αυτό το πρόβλημα, υπάρχει η δυνατότητα της σειριοποίησης των δεδομένων (serialization) και της αποθήκευσής τους σε αρχεία μεσαίου μεγέθους (κάποιων MB) τα οποία μπορούν να αναγνωστούν γραμμικά.

Το φορμάτ TFRecord είναι ένα φορμάτ που επιτρέπει την αποθήκευση σειράς δυαδικών εγγραφών. Διαβάστε σχετικά για το TFRecord and tf.Example και tf.data: Build TensorFlow input pipelines.

Σημειώστε ότι με τη μέθοδο αυτή θα πρέπει να γίνει import η `tensorflow_datasets` και να χρησιμοποιήσουμε την `tfds.load` ώστε να αποθηκευθεί το σύνολο δεδομένων σε αρχεία tfrecord στο δίσκο (δείτε εδώ ένα παράδειγμα). Φυσικά μπορούμε να μετατρέψουμε και τα πρωτογενή δεδομένα (raw data) του dataset όπως αρχεία jpg σε φορματ tfrecord όπως εδώ.