UNIVERSITATEA ”ALEXANDRU-IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ

Fashion Detect Net

propusă de

Elena Cătălina Kissinger

Sesiunea: februarie, 2024

Coordonator științific

Conf. Dr. Anca Ignat

UNIVERSITATEA ”ALEXANDRU-IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

Fashion Detect Net

Elena Cătălina Kissinger

Sesiunea: februarie, 2024

Coordonator științific

Conf. Dr. Anca Ignat

Cuprins

Motivație

Optimizarea proceselor și eficiența operațională în domeniul modei depind de automatizarea sortării și a categorizării. O soluție tehnică inovatoare este oferită pentru a gestiona dificultățile de gestionare a stocurilor din sectorul modei prin implementarea algoritmului StyleDetectNet.

Cu atât de multe articole și colecții diferite într-o lume care se schimbă rapid, gestionarea inventarului devine o treabă mai dificilă. Companiile de modă ar putea include algoritmul sugerat în sistemele lor de inventar pentru a recunoaște și clasifica automat fiecare produs în funcție de atributele și stilul său unic. Acest lucru reduce drastic cantitatea de muncă umană necesară pentru sortare și scurtează dramatic timpul necesar pentru actualizarea și menținerea inventarului.

Sectorul modei poate beneficia de o gestionare semnificativ îmbunătățită a stocurilor, de distribuția precisă a produselor și de riscuri reduse legate de epuizări și suprastoc prin utilizarea acestui algoritm. Cu capacitatea de a eficientiza procedurile logistice, această tehnologie care schimbă jocul ar putea economisi o sumă semnificativă de bani și timp.

Concomitent, acest algoritm, care este o componentă fundamentală, poate deschide ușa pentru crearea de aplicații inovatoare menite să asiste procesul de design de modă. Proiectul StyleDetectNet poate oferi o bază solidă pentru crearea unei varietăți de aplicații legate de modă, oferind sugestii personalizate pentru ansambluri, nuanțe și tendințe de modă. Motivele enumerate mai jos pot determina extinderea proiectului:

Îmbunătățirea experienței de cumpărături online: Adăugarea algoritmului StyleDetectNet la o aplicație de cumpărături online are potențialul de a îmbunătăți experiența de cumpărături online pentru utilizatori. Dacă se adaugă un algoritm în care sistemul are capacitatea de a evalua propriile preferințe de stil ale clienților și de a oferi recomandări de ținute personalizate, atunci se accelerează procesul de selecție și sporește fericirea utilizatorilor. Sugestii pentru culorile îmbrăcămintei:

Sugestii de ținută pentru evenimente speciale: Sfaturi bazate pe evenimente: se poate creea o aplicație, care poate oferi recomandări de garderobă specifice întâlnirilor formale, întâlnirilor ocazionale sau evenimentelor sportive,aplicația având la bază StyleDetectNet. Acest lucru ar face alegerea hainelor mai ușoară, având în vedere particularitățile ocaziei.

Introducere

Proiectul ”StyleDetectNet” reprezintă o inițiativă ce se bazează pe tehnologii avansate pentru a aduce inovații în domeniul recunoașterii imaginilor de modă. Proiectul este scris în Python și folosește biblioteci precum OpenCV, TensorFlow, Keras și NumPy. Tehnica sa principală este o abordare metodică a curățării datelor, care garantează un set de date de înaltă calitate prin eliminarea imaginilor redundante, concentrându-se pe recunoașterea si diferețierea fiecărei categorii , respectiv sugategorii și al altor atribute relavante.

În domeniul procesării imaginilor, proiectul urmează linii teoretice care utilizează matrice NumPy pentru a standardiza imaginile într-un format de 80x60 pixeli. Etichetele sunt codificate folosind metode precum codificarea one-hot pentru a facilita antrenamentul eficient al modelului.

Rețelele neurale convoluționale (CNN) sunt o componentă cheie în cadrului teoretic și practic al proiectului. Capacitatea CNN-urilor de a extrage automat informații ierarhice din imagini este motivul pricipal din spatele alegerii mele al acestor rețele . Designul modular al CNN-urilor (Convolutional Neural Network) în cadrul proiectului subliniază angajamentul său teoretic față de adaptare și scalabilitate.

Pentru a îmbunătăți generalizarea modelului, proiectul adaugă abordări de creștere a setului de date prin augumentarea datelor. Setul de date de antrenament este diversificat artificial prin modificări cum ar fi rotația, deplasarea și răsturnarea, care permit modelului să gestioneze cu eficacitate diferențele de imagine din lumea reală. În plus, modelele pre-antrenate sunt utilizate în integrarea teoretică a învățării prin transfer, ceea ce îmbunătățește capacitatea modelului de a identifica caracteristici complicate cu o cantitate relativ mică de date etichetate.

Lucrarea are ca obiectiv principal clasificarea eficientă a articolelor de îmbrăcăminte și încălțăminte în imagini, oferind astfel o soluție automatizată pentru identificarea și catalogarea acestora.

Această lucrare prezintă o clasificare de imaginii a hainelor, în Capitolul 1 se vor găsi elemente teoretice despre tehnicile folosite. În capitolul 2 va oferi o explicație detaliată a metodologiei adoptate în procesul de dezvoltare a sistemului. Se va face o trecere complexa prin etapele specifice, precum curățarea și preprocesarea datelor, detaliind, de asemenea, elaborarea particularităților distinctive ale proiectului. Această secțiune va servi drept ghid general pentru înțelegerea modului în care am luat deciziile în ceea ce privește implementarea și configurarea rețelei neurale convoluționale.

În Capitolul 3, vor fi prezentate detaliile experimentelor și rezultatele obținute prin aplicarea metodologiei dezvoltate. Acest capitol va constitui o analiză detaliată a performanțelor sistemului în recunoașterea imaginilor din industria modei. Rezultatele obținute vor fi sintetizate și interpretate într-un mod logic, oferind o perspectivă clară asupra eficacității sistemului propus.

Contribuții

Contribuțiile mele la acest proiect au constat în proiectarea și implementarea algoritmilor, colectarea și pregătirea datelor, evaluarea performanței modelelor, analiza rezultatelor și redactarea documentației aferente. Am avut grijă ca algoritmii să fie eficienți, am asigurat calitatea datelor folosite și am evaluat performanța modelelor în mod corespunzător. În plus, am contribuit la analiza datelor și la formularea concluziilor relevante.

Capitolul 1

Descrierea noțiunilor teoretice

1.1 Rețele neurale convoluționale

În domeniul inteligenței artificiale, rețelele neurale convoluționale (CNN) au devenit o paradigmă revoluționară, în special în domeniul procesării imaginilor. Aceste rețele neurale sunt deosebit de potrivite pentru sarcini precum detectarea obiectelor, recunoașterea modelelor și clasificarea imaginilor, datorită capacității lor de a învăța automat și adaptiv reprezentările ierarhice ale intrărilor.

Convoluția, un proces matematic care combină integrarea datelor de intrare cu filtre sau nuclee care pot fi învățate, este ideea fundamentală din spatele CNN-urilor. Rețeaua poate identifica ierarhii spațiale și poate extrage caracteristici utile din fotografii datorită acestei proceduri. Straturile de grupare (Pooling Layers) sunt utilizate împreună cu straturile convoluționale (Convolutional Layers) pentru a reduce complexitatea de calcul, menținând în același timp caracteristicile cheie prin subeșantionarea dimensiunilor spațiale ale intrării.

Structura tipică a rețelelor neurale convoluționale (CNN) cuprinde, de obicei, straturi convoluționale, de grupare și complet conectate, dispuse într-un mod stratificat. Modelele și caracteristicile locale sunt înregistrate de straturile convoluționale, în timp ce invarianța de translație este încurajată de straturile de grupare, care scad dimensiunile spațiale. La sfârșitul rețelei, straturile complet conectate combină funcțiile recuperate pentru a face alegerea finală.

Straturi convoluționale: CNN-urile au la bază straturi convoluționale. În aceste rețele se recurge, de obicei, la utilizarea filtrelor sau nucleelor, entități cu capacitatea

de adaptare în timpul antrenării, în scopul aplicării operațiunilor de convoluție asupra datelor de intrare. Aceste filtre detectează modele, margini și texturi locale trecând peste dimensiunile spațiale introduse. Filtrele multiple ale unui strat convoluțional permit rețelei să învețe o varietate de caracteristici.

Într-o operație de convoluție, filtrul (sau nucleul) este aplicat asupra datelor de intrare prin înmulțirea element cu element și însumarea rezultatelor pentru a produce hărți de caracteristici. Acest proces ajută la captarea ierarhiilor spațiale din datele de intrare, permițând rețelei neurale să identifice și să înțeleagă trăsături semnificative în imagini.

Straturi de grupare: Straturile convoluționale sunt precedate de straturi de grupare. Ele păstrează invarianța translațională reducând în același timp dimensiunile spațiale ale hărților de caracteristici pentru a se concentra asupra celor mai importante date. O metodă populară numită „max pooling" păstrează valoarea maximă într-o anumită regiune în timp ce elimină datele mai puțin importante. Această fază de subeșantionare scade șansa de supraadaptare, îmbunătățind în același timp eficiența de calcul.

Fully Connected Layers: Straturile complet conectate din arhitectura rețelelor neurale convoluționale funcționează cu hărți de caracteristici care au fost aplatizate de nivelele anterioare. În aceste straturi, neuronii sunt conectați la neuronul fiecărui alt strat, emulând structura unei rețele neuronale convenționale. Pentru a ajunge la o judecată finală, straturile complet conectate integrează caracteristici de nivel înalt care au fost învățate de straturile anterioare. Aceste straturi oferă frecvent probabilități de ieșire pentru diferite clase în aplicațiile de clasificare a imaginilor.

În contextul rețelelor neuronale, cele două forme principale de învățare sunt: supervizată, nesupervizată.

Etichetele care sunt prealocate fiecărei intrări sunt esențiale pentru învățarea supervizată. Aceste clasificări servesc în esență ca obiective. Discrepanța dintre ele și previziunile rețelei indică inexactitatea. Legătura dintre intrare, sau etichetă și ieșire, sau precizia modelului antrenat, poate fi măsurată folosind funcția de cost.

Învățarea nesupervizată diferă de învățarea supervizată prin faptul că etichetele nu sunt folosite. Acest tip de învățare adună date în categorii discrete (cum ar fi gruparea) pe care o persoană nu le-ar putea recunoaște.

1.2 Augumentarea datelor

Mărirea seturilor de datelor joacă un rol esențial în îmbunătățirea performanței și robusteței rețelelor neurale, în special în domeniul vederii computerizate.

Prin utilizarea acestei strategii, setul de date este mărit și intrarea pe care modelul le vede în timpul antrenamentului este diversificată prin aplicarea diferitelor transformări datelor deja existente. Teoretic, creșterea datelor se bazează pe capacitatea sa de a reduce supraadaptarea, de a îmbunătăți generalizarea și de a ușura adaptabilitatea modelului la diverse situații din lumea reală.

Pentru a depăși constrângerile impuse de un set de date finit, creșterea datelor este un instrument crucial. Obținerea unui set de date extins și divers poate implica adesea costuri considerabile. O abordare viabilă a acestei probleme este mărirea setului de date, care utilizează artificial transformări precum rotația, răsturnarea, mărirea și deplasarea. Datele de antrenament devin variabile ca urmare a acestor modificări, ceea ce descurajează modelul să învețe anumite instanțe și, în schimb, îl împinge să învețe caracteristici invariante.

1.3 Normalizarea datelor

Normalizarea datelor este o etapă crucială în preprocesarea datelor pentru CNN-uri. Această tehnică este utilizată pentru a asigura că datele de intrare au o distribuție uniformă și pentru a facilita convergența rapidă a modelului în timpul antrenării. La nivel conceptual, normalizarea datelor înseamnă aducerea datelor la o scală comună, eliminând astfel discrepanțele în variația valorilor. În cazul imaginilor, aceasta înseamnă aducerea intensității pixelilor la aceeași scală, astfel încât nu există diferențe semnificative între valorile acestora. Această uniformizare a datelor ajută algoritmul să învețe mai eficient și să generalizeze mai bine pe datele de test.

Pentru CNN-uri, normalizarea se realizează adesea folosind metoda StandardScaler, care calculează media și deviația standard a datelor și apoi le aplică o transformare pentru a le centra în jurul unei medii de zero și a le asigura o deviație standard de unu. Acest lucru asigură că datele au o distribuție normală, ceea ce poate facilita procesul de învățare al modelului.

Beneficiile normalizării datelor pentru CNN-uri sunt multiple. În primul rând, aceasta poate ajuta la evitarea fenomenului de explozie a gradientului, care poate

apărea în timpul antrenării și poate împiedica convergența modelului. În al doilea rând, normalizarea poate duce la o convergență mai rapidă a modelului, deoarece parametrii acestuia sunt actualizați mai eficient în timpul antrenării. În plus, poate reduce sensibilitatea rețelei la mărimea absolută a datelor de intrare, ceea ce poate îmbunătăți generalizarea modelului la datele noi și nevăzute.

1.4 Rețele neurale convoluționale pre-antrenate

În domeniul rețelelor neuronale convoluționale (CNN), MobileNetV2 și VGG16 sunt două arhitecturi proeminente cu caracteristici și utilizări distincte.

MobileNetV2 Scopul MobileNetV2 este de a rezolva dificultățile de calcul implicate în implementarea rețelelor neuronale profunde pe dispozitive cu resurse limitate, în special dispozitive mobile și edge. Google a lansat MobileNetV2 în 2018, subliniind eficiența fără a sacrifica performanța. Utilizarea reziduurilor inversate cu blocaje liniare, care are ca rezultat o arhitectură ușoară, dar puternică, este o caracteristică unică. Un element crucial sunt circumvoluțiile separabile în profunzime, care reduc dramatic numărul de parametri și cheltuielile de calcul fără a sacrifica expresivitatea. Din acest motiv, MobileNetV2 este potrivit pentru aplicațiile în timp real care rulează pe sisteme încorporate care au putere de procesare scăzută.

VGG16 Grupul de geometrie vizuală de la Universitatea din Oxford a venit cu ideea pentru VGG16. Provocarea de recunoaștere vizuală la scară largă ImageNet din 2014 a ajutat-o să devină binecunoscută. Cu arhitectura sa consistentă și 16 straturi de greutate - 13 convoluționale și trei complet conectate - VGG16 se distinge prin simplitatea sa. Cu straturi convoluționale care includ filtre minuscule 3x3, rețeaua VGG16 este bine cunoscută pentru arhitectura sa profundă și omogenă, care permite rețelei să învețe caracteristici ierarhice complexe. În ciuda faptului că are o arhitectură simplă, VGG16 a fost folosit ca model de referință pentru o serie de sarcini de clasificare a imaginilor și a pus bazele pentru proiecte mai complexe în viitor.

Comparatii Cazurile de utilizare și filozofiile lor arhitecturale sunt cele în care MobileNetV2 și VGG16 diferă cel mai mult. MobileNetV2 este o opțiune excelentă pentru edge computing și aplicații mobile, deoarece acordă o prioritate ridicată eficienței

și este proiectat pentru a fi implementat pe dispozitive cu resurse limitate. Cu toate acestea, VGG16 este o opțiune puternică pentru sarcinile de clasificare a imaginilor, deoarece se concentrează pe învățarea caracteristicilor ierarhice și de profunzime, mai ales atunci când resursele de calcul nu sunt la fel de limitate.

Capitolul 2

Metodeologie

2.1 Tehnologii și biblioteci folosite

Proiectul „StyleDetectNet" este o investigație în domeniul recunoașterii imaginilor și folosește o gamă de biblioteci specializate. Tehnologiile și bibliotecile utilizate în acest proiect vor fi tratate în detaliu în această secțiune.

Biblioteca Pandas este utilă atunci când vine vorba de manipularea datelor de intrare, care este un aspect crucial al proiectului. Funcțiile puternice oferite de Pandas permit citirea și preprocesarea eficientă a setului de date. Proiectul se concentrează pe o colecție de fotografii legate de diferite articole de modă, cum ar fi șlapi, sandale și pantofi, bluze,rochii etc. Datele sunt organizate într-un fișier CSV numit styles.csv, iar Pandas este folosit pentru a le filtra și gestiona eficient.

Proiectul folosește OpenCV (Open Source Computer Vision Library), o bibliotecă de referință în domeniul procesării imaginilor, pentru manipularea imaginilor. OpenCV oferă o gamă largă de algoritmi de manipulare și analiză a imaginilor, inclusiv capacitatea de a redimensiona imaginile la dimensiunile dorite. În rețeaua neurală, această preprocesare minuțioasă este esențială pentru obținerea unor rezultate precise.

Preprocesarea setului de date folosește funcții din biblioteca scikit-learn. Funcțiile folosite în soluție sunt `train_test_split` și `StandardScaler` pentru împărțirea, respectiv normalizarea datelor.

Componenta centrală a proiectului este implementarea rețelelor CNN. Folosind biblioteca TensorFlow și modulul Keras, arhitectura modelelor este definită în detaliu în codul sursă. Straturile convoluționale, de grupare și complet conectate formează structura stratificată a modelului. Aceste straturi sunt necesare pentru extragerea auto-

mată a informațiilor ierarhice și pentru captarea ierarhiilor spațiale în imaginile de intrare.

Metrici precum acuratețea sunt folosite în proiect pentru a evalua performanța modelului. Biblioteca Matplotlib este folosită pentru a calcula și reprezenta grafic aceste valori într-o manieră intuitivă. Graficele rezultate fac posibilă urmărirea și examinarea modului în care performanța modelului se modifică în timpul antrenamentului.

```
plt.plot( *args: history.history['accuracy'], label='Training Accuracy')
if 'val_accuracy' in history.history:
    plt.plot( *args: history.history['val_accuracy'], label='Validation Accuracy')
plt.title(f'Model Accuracy(articleType){name} - {epochs} epochs, optimizer:{nameopt}')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

Figura 2.1: Generarea graficelor de acuratețe

2.2 Setul de date

Setul de date este compus din aproximativ 44.000 de imagini cu articole vestimentare și un fișier csv care conține etichetele fiecărei imagini. Fiecare produs are un ID unic, cum ar fi 42431. Astfel putem prelua din fișierul styles.csv imaginea produsului gasită la adresa ../images/42315.jpg. Etichetele găsite în csv sunt urmatoarele: masterCategory, subCategory, articleType, baseColour, season, year, usage, și productDisplayName. În solutia mea clasific imaginiile în funcție de etichetele: masterCategory, subCategory, articleType, season și usage. Mai jos o sa explic cu am imparțit setul de date și cum am făcut preprocesare datelor în funcție de etichetele prezente și de nivelul de diversitate al imaginilor.

Master Category Pentru accestă categorie datele au fost împarțite. În cadrul acestei coloane se găsesc numeroase etichete, dintre care au fost selectate următoarele: „Apparel”, „Accessories”, „Footwear”, iar restul etichetelor au fost transformate în etichta ”Others”. Aceasta procesare a datelor este rezultatul inconsitentei setului de date dupa cum puteți observa in figura 2.2.

Acest proces de pregătire a datelor implică patru funcții.

Prima, `modify_csv()`, ajustează coloana *masterCategory* din fișierul CSV original, înlocuind toate categoriile cu ”Others” exceptând, „Footwear” pentru ca ulterior

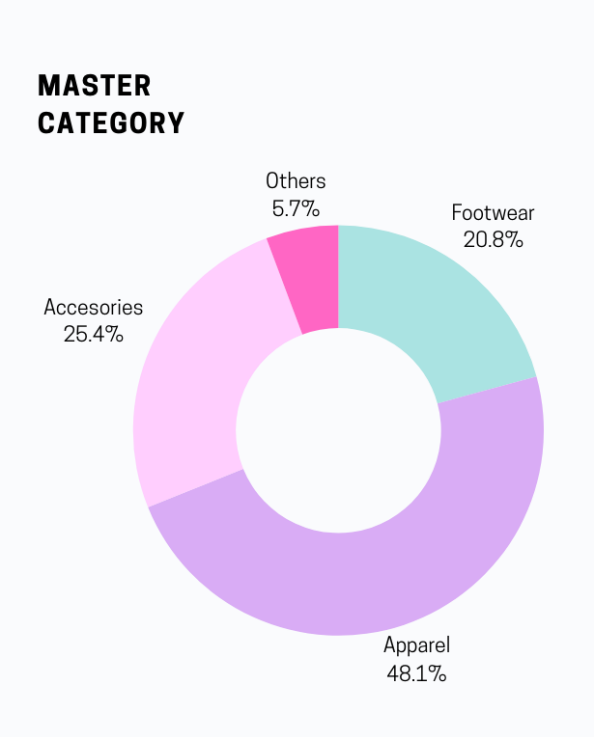

Figura 2.2: Master Category: distribuția datelor

```
def modify_csv():
    df = pd.read_csv('data/myntradataset_original/styles.csv')
    df['masterCategory'] = df['masterCategory'].apply(lambda x: x if x == 'Footwear' else 'Others')
    df.to_csv( path_or_buf: 'data/myntradataset_masterCategory/masterCategory_filtrat.csv', index=False)
    count_footwear = df['masterCategory'].value_counts()['Footwear']
```

Figura 2.3: Funcția modify_csv

să facem o clasificare pentru „Apparel” și „Accessories” și una pentru „Footwear” și ”Others”. Acestă împartire a fost facută ca modelele să aibă o performanța mai bună.

```
def delete_invalid_images():
    df = pd.read_csv('data/myntradataset_masterCategory/styles.csv')
    valid_categories = ['Apparel', 'Accessories']
    folder_path = 'data/myntradataset_masterCategory/images'
    for filename in os.listdir(folder_path):
        img_name = os.path.splitext(filename)[0]
        img_id = int(img_name)
        if img_id in df['id'].values:
            img_category = df.loc[df['id'] == img_id, 'masterCategory'].values[0]
            if img_category not in valid_categories:
                img_to_delete = os.path.join(folder_path, filename)
                os.remove(img_to_delete)
                print(f'Deleted image: {img_to_delete}')

    df_filtered = df[df['masterCategory'].isin(['Apparel', 'Accessories'])]
    df_filtered.to_csv('masterCategory_filtrat.csv', index=False)
```

Figura 2.4: Funcția delete_invalid_images

Cum puteți observa in figura 2.4 funcția `delete_invalid_images` elimină imaginile care nu se încadrează în categoriile ”Apparel” sau ”Accessories”, asigurând ast-

fel integritatea setului de date. Funcția `load_and_process_images` încarcă și prelucrează imaginile din fișierul filtrat, aplicând aceeași redimensionare și normalizare a pixelilor. Prin aceste procese riguroase, asigurăm că setul nostru de date rămâne concentrat și de înaltă calitate, pregătit pentru analiza și explorarea ulterioară.

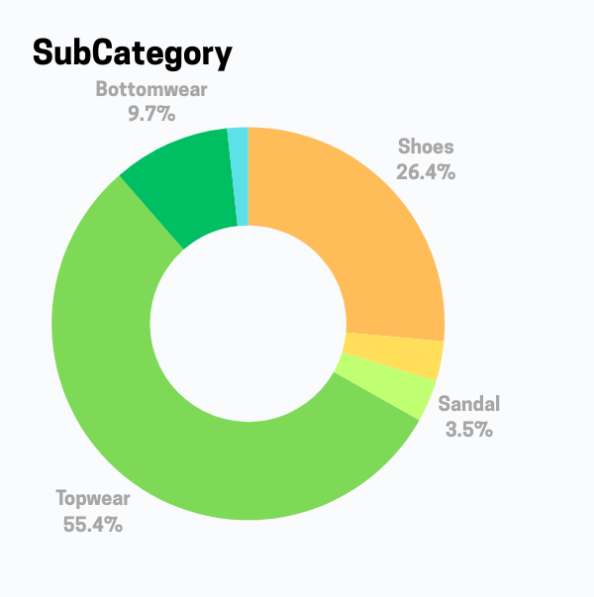

Figura 2.5: Subcategory: distribuția datelor

Subcategory În acesta coloana se găsesc 45 de atribute unice, din care selectam 8, respectiv 'Shoes', 'Flip Flops', 'Sandal', 'Topwear', 'Bottomwear', 'Dress'. Procesarea datelor constă în ștergerea imaginiilor si etichetelor nefolositoare, redimensionare imaginiilor pe 80x60 pixeli și codificarea etichetelor rămase în csv cu tehnica one-hot encoding. Aceasta presupune transoformarea fiecărei etichete într-un vector binar, astfel încât fiecare categorie să fie reprezentată de o valoare binară distinctă.

Article Type Categoria *Article Type* conține 142 de atribute unice. Asemănător cu celelate categorii și aici se impart datele în două subseturi, respectiv încălțaminte și haine, astfel ele mai departe în soluție sunt clasificate separat. Din cele 142 de atribute sunt selectate urmatoarele: 'Shirts', 'Blazers', 'Jeans', 'Sweatshirts', 'Dresses', 'Shorts', 'Trousers', 'Skirts' pentru haine, iar pentru încălțaminte 'Casual Shoes', 'Flip Flops', 'Sandals', 'Formal Shoes', 'Casual Shoes', 'Flats', Sports Shoes', 'Heels'.

Cu toate acestea, procesarea datelor pentru setul de haine diferă de procesarea celui de încălțaminte. Astfel, eticheta *"Shirts"* din setul de date este ierarhic structurată, cuprinzând subcategorii distincte precum "Tops" și "T-shirts".

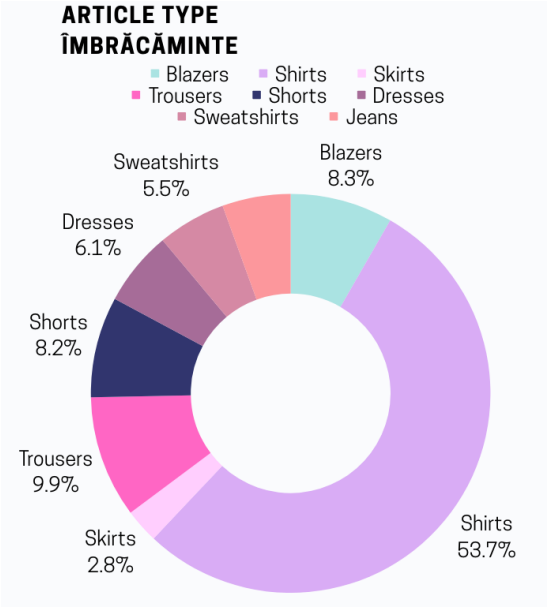

Figura 2.6: Subcategory: distribuția hainelor

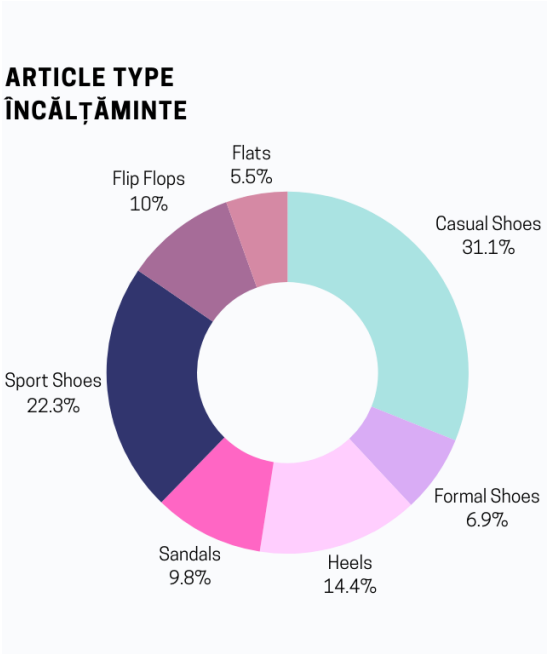

Figura 2.7: Article Type: distribuția încălțămintii

Această structură ierarhică este folosită pentru o clasificare detaliată a articolelor vestimentare, unde ”Shirts” reprezintă o categorie generală, iar ”Tops” și ”T-shirts” reprezintă subcategorii specifice. Astfel, această abordare oferă o organizare semantică mai fină, facilitând înțelegerea și analiza diferitelor tipuri de articole vestimentare. Puteți observa în figura 2.8 distribuția datelor.

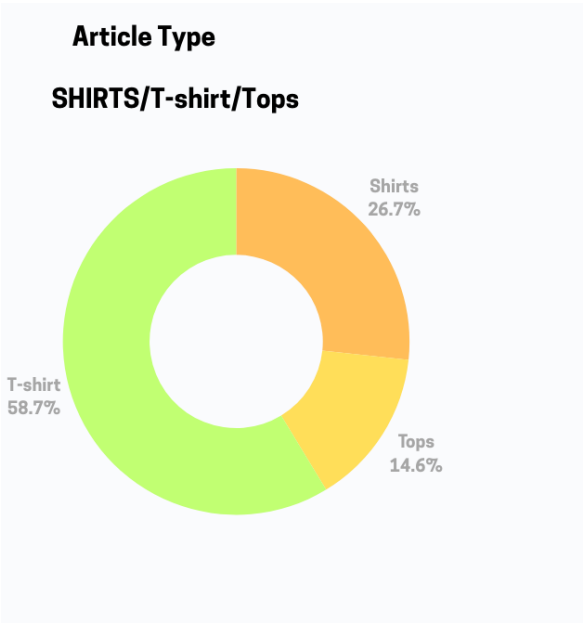

Figura 2.8: Distribuția ”Shirts”, ”Tops” și ”T-shirts”

Style/Usage Categoria *Usage* conține 8 de atribute unice, dintre care selectăm 3, respectiv 'Sports', 'Formal', 'Casual'. Puteți observa distributia datelor în figura 2.9.

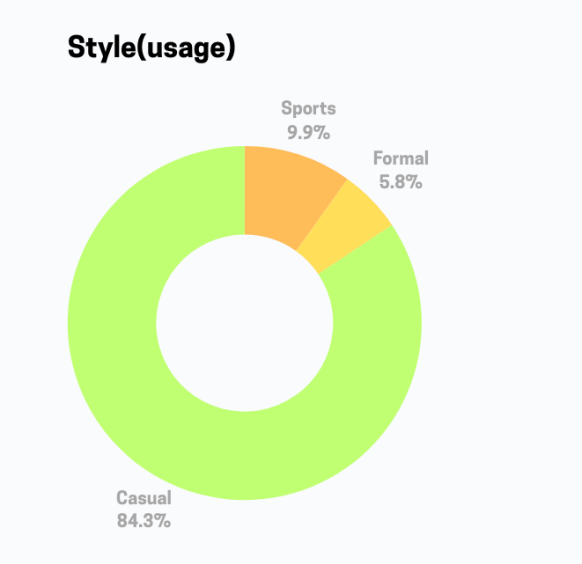

Figura 2.9: Style: distribuția datelor

Season Inițial, setul de date conținea patru etichete distincte pentru sezon: "Summer", "Spring", "Winter" și "Fall". Totuși, în timpul analizei datelor, am observat că aceste etichete pot fi interpretate în mai multe moduri, iar clasificarea lor poate fi subiectivă sau dificil de realizat în mod precis.

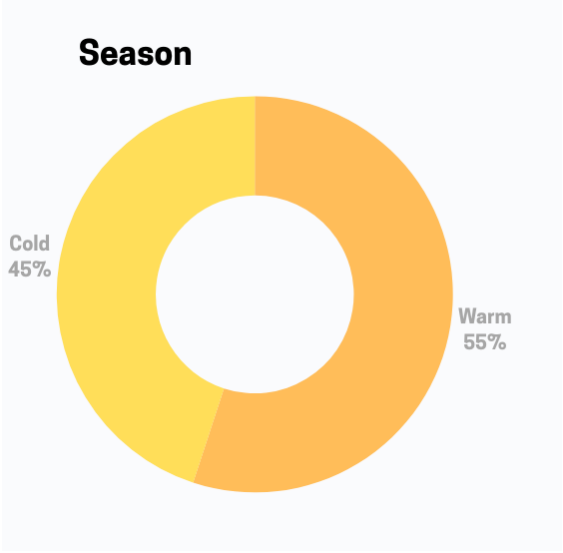

Figura 2.10: Season: distribuția datelor

Pentru a simplifica procesul de clasificare și pentru a crea categorii mai generale și mai ușor de interpretat, am decis să efectuăm o mapare a acestor etichete în două ca-

tegorii principale: ”warm” pentru vară și primăvară și ”cold” pentru iarnă și toamnă.

Împartirea datelor Metoda `train_test_split` din biblioteca scikit-learn este utilizată pentru a împărți setul de date în două subseturi: un set de antrenare și un set de validare. Această metodă primește ca parametri images și labels, care reprezintă imaginile și etichetele corespunzătoare din setul de date. Parametrul test_size specifică proporția de date care vor fi alocate pentru setul de validare, în acest caz 20%. Prin setarea random_state la o anumită valoare, în cazul nostru 42, asigurăm reproducibilitatea împărțirii datelor, adică aceeași împărțire va fi obținută de fiecare dată când rulăm codul.

```
train_images, val_images, train_labels, val_labels = train_test_split( *arrays: images, labels, test_size=0.2,
                                                                        random_state=42)
```

Figura 2.11: Împarțirea setului de date

Augumentarea În dezvoltarea algoritmilor de învățare automată, augmentarea datelor este o tehnică esențială pentru îmbunătățirea generalizării și a performanței modelului. Această tehnică implică generarea de noi exemple de antrenare prin aplicarea unor transformări aleatorii și perturbări asupra datelor existente, astfel încât să obținem o varietate mai mare de exemple, fără a adăuga manual noi date.

În figura 2.11 se poate observa că în codul dat se folosește funcția din biblioteca Keras `ImageDataGenerator`cu ajutorul căreia datele sunt augmentate prin aplicarea unor transformări precum rotație, deplasare pe orizontală și verticală, deformări aleatorii, zoom, răsturnare orizontală, completare a pixelilor, modificare a luminozității, schimbare aleatorie a valorilor canalelor de culoare și tăiere aleatoare a imaginilor.

```
data_gen = ImageDataGenerator(
    rotation_range=40,  # Rotirea imaginii într-un interval de 40 de grade
    width_shift_range=0.25,  # Deplasarea imaginii pe orizontală cu maximum 20%
    height_shift_range=0.25,  # Deplasarea imaginii pe verticală cu maximum 20%
    shear_range=0.25,  # Deformatii aleatorii ale imaginii cu maximum 20%
    zoom_range=0.3,  # Zoom aleatoriu în interiorul imaginii cu maximum 20%
    horizontal_flip=True,  # Răsturnare orizontală aleatoare a imaginii
    fill_mode='nearest',  # Modul de completare a pixelilor după transformări
    brightness_range=[0.8, 1.2],  # Modificarea luminozitătii între 80% și 120%
    channel_shift_range=20,  # Schimbarea aleatorie a valorilor canalelor de culoare
    featurewise_center=True,  # Tăiere aleatoare a imaginilor

)
```

Figura 2.12: Augumentarea datelor

Aceste transformări ajută modelul să devină mai robust și mai capabil să generalizeze mai bine pe datele noi sau mai puțin cunoscute. Prin aplicarea augmentării datelor, putem evita supraadaptarea și îmbunătăți performanța generală a modelului în timpul testării pe date reale.

Normalizarea datelor este un proces esențial în preprocesarea datelor în contextul antrenării modelelor de rețele neurale. Prin normalizare, se urmărește aducerea datelor la o scară comună și uniformizarea distribuției acestora. Acest lucru este realizat, de obicei, folosind metode precum StandardScaler, care centrează datele în jurul mediei zero și le ajustează pentru a avea o deviație standard de 1.

```
scaler = StandardScaler()
train_images_scaled = scaler.fit_transform(train_images.reshape(train_images.shape[0], -1)).reshape(
    train_images.shape)
val_images_scaled = scaler.transform(val_images.reshape(val_images.shape[0], -1)).reshape(val_images.shape)
```

Figura 2.13: Normalizarea datelor

Normalizarea datelor a fost folosită pentru a elimina discrepanțele între caracteristici, astfel încât toate caracteristicile să aibă o importanță egală în procesul de învățare al modelului. Aceasta contribuie la convergența mai rapidă și mai stabilă a modelului, reducând timpul necesar pentru antrenare și riscul de supraantrenare.

De asemenea,în timpul antrenării am întâmpinat niște valori extreme, dar dupa normalizarea datelor s-a îmbunătățit stabilitatea antrenării, făcând modelul mai puțin sensibil. Aceasta contribuie la generalizarea mai bună a modelului, adică capacitatea sa de a face predicții precise și consistente pe date noi sau nevăzute.

2.3 Soluția problemei

Soluția implementată pentru clasificarea imaginilor a implicat mai mulți pași cheie. Inițial, setul de date a suferit o preprocesare meticuloasă, inclusiv eliminarea imaginilor irelevante și clasificarea datelor în clase adecvate. Acest pas a asigurat că setul de date rămâne concentrat și aliniat cu obiectivele sarcinii.

Ulterior, o arhitectură de rețea neuronală convoluțională (CNN) a fost proiectată și implementată folosind Keras. Arhitectura modelului cuprindea straturi de intrare, ascunse și de ieșire, fiecare strat servind o funcție specifică în procesul de extracție și clasificare a caracteristicilor.

În căutarea îmbunătățirii performanței modelului, a fost efectuată o explorare amănunțită a algoritmilor de optimizare, cu un accent deosebit pe optimizatorii Adam și RMSprop. Optimizatorii au fost aleși pentru că, în urma mai multor rulări, s-a constatat că aceștia au furnizat cele mai bune rezultate.

2.3.1 BinaryOutputModel

Această implementare Keras a unei rețele neurale convoluționale (CNN) este adaptată pentru clasificarea imaginilor binare, discernând între categoriile „Apparel" și „Accessories". Din punct de vedere structural, modelul cuprinde diverse straturi concepute strategic pentru a procesa și extrage caracteristici din imaginile de intrare.

```
inputs = Input(shape=(height, width, 3))
# Build hidden layers
hidden_layers = self.make_default_hidden_layers(inputs)
```

Figura 2.14: Definirea stratului de intrare

În centrul său se află stratul de intrare, definind dimensiunile imaginilor de intrare ca (înălțime, lățime, 3), unde înălțimea și lățimea reprezintă dimensiunile imaginii, iar 3 înseamnă numărul de canale de culoare (RGB). Acest strat servește ca poartă de intrare pentru datele de imagine în arhitectura modelului. Straturile ascunse ale modelului sunt realizate în cadrul funcției `make_default_hidden_layers` și constituie coloana vertebrală a extragerii caracteristicilor. Se folosesc straturile Conv2D, aceste segmente efectuează operații de convoluție pe imaginile de intrare, detectând cu abilități modele și caracteristici complicate. Pentru o înțelegere detaliată a configurației standard a straturilor ascunse utilizate în soluție,putem vedea Figura 2.9.

```
def make_default_hidden_layers(self, inputs):
    x = Conv2D( filters: 32, kernel_size: (3, 3), activation="relu", padding="valid", kernel_initializer=glorot_uniform(seed=42))(inputs)
    x = MaxPooling2D(pool_size=(2, 2), strides=(1, 1))(x)
    x = BatchNormalization()(x)
    x = Dropout(0.25)(x)
    x = Conv2D( filters: 16, kernel_size: (3, 3), activation="relu", padding="valid")(x)
    x = MaxPooling2D(pool_size=(2, 2), strides=(1, 1))(x)
    x = BatchNormalization()(x)
    x = Dropout(0.25)(x)
    x = Flatten()(x)
    x = Dense( units: 64, activation="relu")(x)
    x = BatchNormalization()(x)
    x = Dropout(0.25)(x)#0.5
    x = Dense( units: 32, activation="relu")(x)
    return x
```

Figura 2.15: Configurare straturilor ascunse pentru BinaryOutputModel

Straturile BatchNormalization au fost adăugate ulterior pentru a ajuta cu preve-

nirea overfitting-ului,astfel, se asigură stabilitatea și eficiența modelului prin normalizarea activărilor și îmbunătățirea performanței.

Trecând în continuare, Flatten Layer remodelează caracteristicile extrase într-o matrice coerentă 1D, pregătindu-le pentru procesarea ulterioară de către straturi dense complet conectate. Aceste straturi dense suplimentare, cuprinzând unități de 64 și respectiv 32, rafinează caracteristicile extrase, pregătindu-le pentru clasificarea finală.

Stratul de ieșire, un singur strat Dens cu 1 unitate cu funcția de activare a sigmoidului, oferă o evaluare probabilistică a clasificării imaginilor de intrare în categoria „Apparel" sau „Accesories". Utilizarea inițializatorului glorot_uniform pentru straturile Conv2D s-a realizat pentru a asigura o inițializare robustă a greutății. În plus, încorporarea regulatorului l2 adaugă un termen de penalizare la funcția de pierdere, reducând efectiv tendințele de overfitting, sporind astfel capacitățile de generalizare ale modelului.

2.3.2 SingleOutputModel

Arhitectura rețelei neurale convoluționale 'SingleOutputModel' este o implementare robustă, adaptată pentru sarcinile de clasificare a imaginilor cu mai multe clase. În esență, modelul folosește o serie de straturi meticulos concepute pentru a distila caracteristici semnificative din imaginile de intrare, permițând în cele din urmă clasificarea precisă în mai multe clase. Acest model a fost folosit pentru algoritmii de clasificare a următoarelor categorii: SubCategory, ArticleType si Usage (Style).

Inițiind cu stratul de intrare, modelul delimitează forma așteptată a imaginilor de intrare, aderând la un format structurat de (înălțime, lățime, 3), unde dimensiunile denotă înălțimea și lățimea imaginii, în timp ce „3" semnifică cele trei canale de culoare corespunzătoare.

Straturile ascunse constituie punctul central al extracției și abstracției caracteristicilor. Folosind o combinație de straturi Conv2D, straturi MaxPooling2D, straturi BatchNormalization și straturi Dropout, aceste straturi funcționează în colaborare pentru a distila modele importante din datele de intrare. . Straturile BatchNormalization asigură stabilizarea activărilor pe diferite straturi, favorizând robustețea modelului. Concomitent, straturile Dropout introduc un mecanism de regularizare, dezactivând aleatoriu o fracțiune de unități în timpul antrenamentului pentru a atenua supraadaptarea, sporind astfel capacitățile de generalizare ale modelului.

```python
class SingleOutputModel:
    1 usage
    def make_default_hidden_layers(self, inputs):
        x = Conv2D( filters: 32, kernel_size: (3, 3), activation="relu", padding="valid", kernel_initializer=glorot_uniform(seed=42))(inputs)
        x = MaxPooling2D(pool_size=(2, 2), strides=(1, 1))(x)
        x = BatchNormalization()(x)
        x = Dropout(0.25)(x)
        x = Conv2D( filters: 16, kernel_size: (3, 3), activation="relu", padding="valid", kernel_regularizer=l2(0.001))(x)
        x = MaxPooling2D(pool_size=(2, 2), strides=(1, 1))(x)
        x = BatchNormalization()(x)
        x = Dropout(0.25)(x)
        x = Flatten()(x)
        x = Dense( units: 64, activation="relu", kernel_regularizer=l2(0.001))(x)
        x = BatchNormalization()(x)
        x = Dropout(0.5)(x)
        x = Dense( units: 32, activation="relu", kernel_regularizer=l2(0.001))(x)
        return x

    def assemble_full_model(self, width, height, num_categories):
        input_shape = (height, width, 3)
        inputs = Input(shape=input_shape)
        # Build hidden layers
        hidden_layers = self.make_default_hidden_layers(inputs)
        # Additional Dense layers
        x = Flatten()(hidden_layers)
        x = Dense( units: 64, activation="relu", kernel_regularizer=l2(0.01))(x)
        x = BatchNormalization()(x)
        x = Dropout(0.5)(x)
        x = Dense( units: 32, activation="relu", kernel_regularizer=l2(0.01))(x)
        category_output = Dense(num_categories, activation='softmax', name='category')(x)
        # Build the model
        model = Model(inputs=inputs, outputs=category_output, name="fashion_category_net")
        return model
```

Figura 2.16: SingleOutputModel

Urmând straturile ascunse, stratul Flatten remodelează meticulos caracteristicile extrase într-o matrice unidimensională, facilitând tranziția fără întreruperi la straturile ulterioare complet conectate (Dense). Aceste straturi dense suplimentare, cuprinzând 64 și respectiv 32 de unități, rafinează meticulos caracteristicile extrase, pregătindu-le pentru etapa finală de clasificare.

Stratul de ieșire, o componentă pivot echipată cu unități num_categories și o funcție de activare softmax.Acest strat organizează procesul de clasificare prin crearea unei distribuții de probabilitate între clasele diferite, indicând șansele ca o imagine să fie asociată cu fiecare categorie în parte. Astfel de rezultate probabilistice împuternicesc procesele de luare a deciziilor prin furnizarea de perspective asupra nivelurilor de încredere asociate cu fiecare clasificare.

2.3.3 VGG16 și MobileNetV2

S-a încercat implementarea unui model propriu pentru a clasifica sezonul, dar obținând rezultate slabe, am determinat explorarea altor opțiuni precum utilizarea modelelor pre-antrenate. VGG și MobileNetV2 sunt două dintre aceste modele pre-antrenate care sunt cunoscute pentru performanța lor în diverse sarcini de clasificare a

imaginilor.

```
def create_pretrained_model(img_shape, num_classes):
    base_model = VGG16(weights='imagenet', include_top=False, input_shape=img_shape)
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense( units: 1024, activation='relu')(x)
    predictions = Dense(num_classes, activation='softmax')(x)
    model = Model(inputs=base_model.input, outputs=predictions)
    return model
```

Figura 2.17: Crearea unui model preantrenat VGG16 pentru clasificarea imaginilor

Cu toate acestea, chiar și după utilizarea acestor modele puternice, rezultatele obținute nu au fost satisfăcătoare. Pentru a îmbunătăți performanța, am încercat o abordare alternativă, asociind sezoanele cu temperatura, astfel rămân două categorii de clasificat. Acest lucru a fost realizat prin definirea unui dicționar care mapează fiecare sezon la un tip de material corespunzător. Am presupus că modelul ar putea fi capabil să facă distincția între hainele de iarnă și celelalte în funcție de grosimea materialului.

```
df = pd.read_csv('data/myntradataset_season/styles.csv')
season_mapping = {'Summer': 'warm', 'Spring': 'warm', 'Winter': 'cold', 'Fall': 'cold'}

# Înlocuirea valorilor din coloana 'season' folosind dicționarul definit anterior
df['season'] = df['season'].replace(season_mapping)
encoded_season = pd.get_dummies(df['season'], prefix='season')
```

Figura 2.18: Mapare sezonului cu valorile specificate

Cu toate acestea, această abordare nu a dus la îmbunătățiri semnificative ale performanței modelului. Problema poate consta în complexitatea datelor sau în faptul că legăturile dintre sezoane și materialele vestimentare sunt mult mai subtile și nu pot fi ușor capturate de modelele de învățare automată.

Totodată, în încercarea de a îmbunătăți performanța modelului meu de clasificare a tipurilor de încălțăminte pentru coloana ArticleType , am optat pentru utilizarea modelelor pre-antrenate. Cu toate acestea, rezultatele obținute nu au fost la înălțimea așteptărilor, iar acest lucru a putut fi influențat de distribuția inechilibrată a datelor în setul meu de antrenare.De exemplu, am constatat că, dintr-un total de aproximativ 9000 de imagini cu încălțăminte, procentsjele apariției etichetelor precum ”SandalS”, ”Heels”, ”Formal shoes” și ”Flip flops” erau relativ mici, cu valori între 6% și 14%.

În concluzie, am experimentat cu modele pre-antrenate pentru a optimiza clasificarea încălțămintei pe categorii de tipuri de articole, însă provocările legate de distribuția dezechilibrată a datelor și de adaptarea modelelor la specificitățile setului meu de date au rămas o barieră în calea obținerii unor rezultate optime.

Capitolul 3

Experimente si rezultate

În acest capitol sunt prezentate detaliile procesului experimental și analiza rezultatelor obținute în cadrul acestui studiu. Această secțiune este crucială pentru a înțelege modului în care modelul a fost construit, antrenat și evaluat, precum și pentru interpretarea performanței acestuia. Prin intermediul experimentelor efectuate și al rezultatelor obținute, vom analiza și evalua eficacitatea și robustețea modelului propus, identificând eventualele puncte tari și slabe și extrăgând concluzii relevante pentru soluția propusă.

3.1 Descrierea predicțiilor și rezultatelor

MasterCategory În cadrul clasificării etichetelor din coloana MasterCategory, procesul este divizat în două etape distincte, fiecare cu scopul său specific. În prima etapă, se realizează o clasificare inițială între două categorii fundamentale: "Apparel" și "Accessories". Acest demers își propune să exploreze posibilitatea de a distinge între articolele de îmbrăcăminte și accesoriile asociate acestora. Odată finalizată această etapă, se trece la următorul nivel de clasificare, care vizează diferențierea între încălțăminte și alte tipuri de articole. În acest context, termenul "others" reprezintă o etichetă introdusă pentru a facilita distincția între articolele de încălțăminte și celelalte atribute incluse în analiză.

În cadrul experimentelor pe algoritmul de clasificare între "Apparel" și "Accessories" , am evaluat performanța doi doptimizatori populari: Adam și RMSprop. Rezultatele obținute au furnizat o imagine clară asupra eficacității și adaptabilității acestor algoritmi în contextul antrenării modelelor noastre de clasificare a imaginilor.

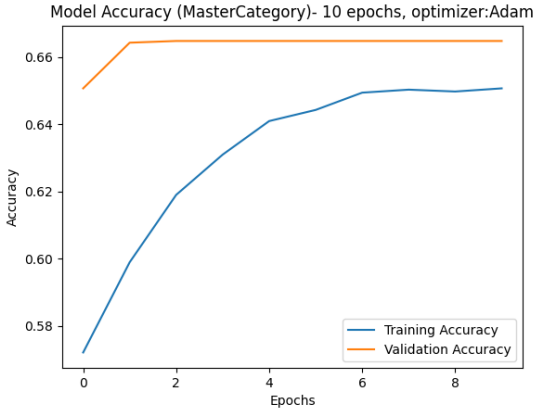

Figura 3.1: Acuratetea modelului pe 10 epoci cu optimizator Adam

Din graficul de mai sus (figura 3.1), care prezinta rezultatele algoritmului cu optimizatorul Adam, putem observa ca datele indică o îmbunătățire a performanței modelului pe setul de antrenare pe măsură ce numărul de epoci crește, cu o creștere a acurateței de la aproximativ 0.58 la 0.65 la epoca 10. Cu toate acestea, pe setul de validare, deși acuratețea începe cu o valoare mai mare decât pe setul de antrenare, fluctuează în jurul aceleiași valori 0.66 . Acest lucru sugerează că modelul este predispus la overfitting, deoarece se adaptează prea bine la datele de antrenare și nu generalizează la datele de validare.

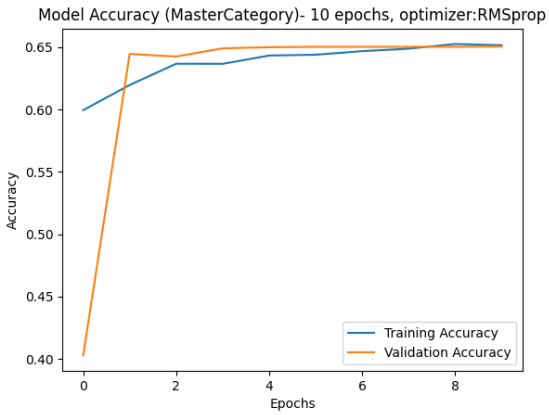

Figura 3.2: Acuratetea modelului pe 10 epoci cu optimizator RmsProp

Pe de altă parte, optimizatorul RMSprop s-a dovedit a fi o alegere competitivă în antrenamentul modelului. Precizia în antrenament pe antrnamentul cu 10 epoci începe ușor mai jos decât cea în validare, situându-se în jurul valorii de 0,40, fluctuând ulterior

între 0,55 și 0,60, atingând aproximativ 0,60 la epoca 10. Pe de altă parte, precizia în validare începe în jurul valorii de 0,60 și fluctuează între 0,60 și 0,65, ajungând la aproximativ 0,65 la epoca 10.

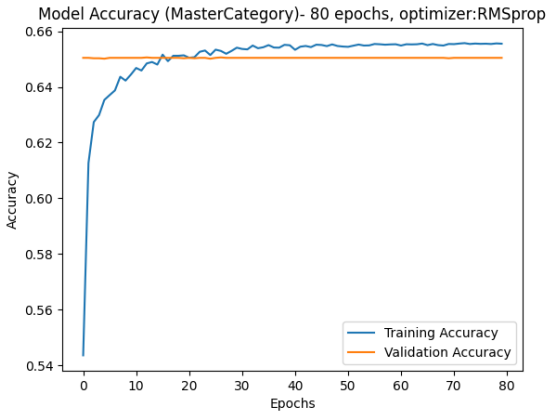

Figura 3.3: Acuratetea modelului pe 80 epoci cu optimizator RmsProp

În schimb pe 80 de epoci, performanța este relativ stabilă pe parcursul antrenamentului. În general, precizia în validare rămâne constantă și relativ ridicată, începând de la 0,65 și atingând un vârf de aproximativ 0,66 la epoca 80. Pe de altă parte, precizia în antrenament începe mai jos, în jurul valorii de 0,54, dar crește pe parcursul antrenamentului, ajungând de la 0,56 la 0,64 la epoca 80. Acest lucru indică faptul că modelul se potrivește bine datelor de antrenament și poate generaliza bine la datele de validare. În ansamblu, precizia modelului rămâne relativ stabilă în timpul antrenamentului, iar precizia în validare depășește în mod constant precizia în antrenament, indicând că modelul generalizează bine către datele nevăzute.

Epochs	**Adam**		**RMSprop**	
	Antrenare	**Validare**	**Antrenare**	**Validare**
10 epoci	64%	66%	65%	65%
80 epoci	66%	64%	66%	64%

Tabela 3.1: Comparație între optimizatorii Adam și RMSprop pentru clasele: ”Apparel” și ”Accessories”

Conform observațiilor prezentate în tabelul 3.1, se constată că acuratețea modelului este relativ scăzută. Această situație este determinată în principal de insuficiența datelor disponibile și de distribuția inechilibrată a acestora. Numărul redus de ima-

gini disponibile pentru categoria principală (MasterCategory), aproximativ 32684, este considerabil redus, iar distribuția acestora nu este uniformă între clase. Această lipsă de date adecvate și distribuția neuniformă contribuie la obținerea unor rezultate sub-optimale, afectând performanța globală a modelului și conducând la o acuratețe mai mică decât cea dorită.

Al doilea algoritm clasifică încalțămintea, la fel ca în algoritmul folosit optimizatorii folosiți sunt Adam si RmsProp.

În figura 3.4 se poate observa cum datele indică că modelul are o performanță promițătoare în timpul antrenamentului. În special, precizia în antrenament începe la aproximativ 0,55 și crește constant pe parcursul epocilor, atingând un vârf de aproximativ 0,75 la epoca 10. Acest lucru sugerează că modelul învață și se adaptează bine la datele de antrenament, obținând o precizie ridicată în final.

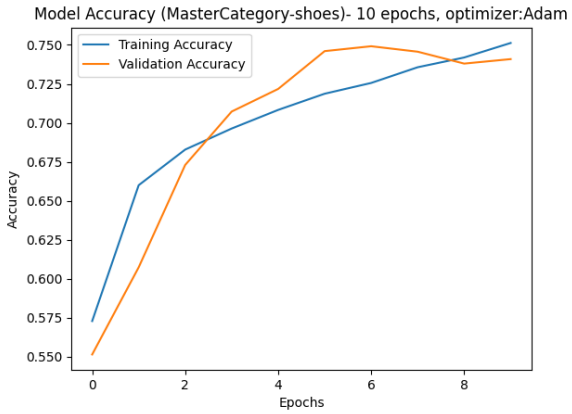

Figura 3.4: Acuratetea modelului pe 10 epoci cu optimizator Adam

În tabela 3.5, se observă diferențele în performanța dintre optimizatorii Adam și RMSprop în diverse epoci. Remarcabil este faptul că ambii optimizatori obțin o precizie mai mare în antrenament și validare atunci când se utilizează Adam. De exemplu, la 10 epoci, Adam atinge o precizie în antrenament de 79.2%, depășind cea a lui RMSprop, care este de 75%, în timp ce precizia în validare pentru Adam este de 78.6%, comparativ cu 72% pentru RMSprop. Similar, la 80 de epoci, Adam menține superioritatea cu o precizie în antrenament de 80%, comparativ cu 80% pentru RMSprop, și o precizie în validare de 79%, în comparație cu 77.5% pentru RMSprop.

În concluzie, rezultatele sugerează că Adam obține în mod constant performanțe superioare față de RMSprop în diferite epoci.

Epochs	Adam		RMSprop	
	Antrenare	**Validare**	**Antrenare**	**Validare**
10 epoci	75%	72%	79,2%	78,6%
80 epoci	80%	79%	80%	77,5%

Tabela 3.2: Comparație între optimizatorii Adam și RMSprop pentru clasele: ”Footwear”, ”Others”

SubCategory Pentru această coloană, am implementat doi algoritmi distincți: unul pentru clasificarea încălțăminți și unul pentru clasificarea îmbrăcăminți. Am optat pentru această abordare pentru a asigura o înțelegere mai profundă și o acuratețe mai mare în ceea ce privește clasificarea fiecărei categorii.

Algoritmul pentru clasificarea încălțăminți operează cu trei clase distincte, respectiv ”Shoes”, ”Flip Flops” și ”Sandals”. În tabelul de mai jos puteți observa rezultatele obținute.

Epochs	Adam		RMSprop	
	Antrenare	**Validare**	**Antrenare**	**Validare**
10 epoci	75%	73%	65%	65%
80 epoci	82%	81%	83%	80%

Tabela 3.3: Comparație între optimizatorii Adam și RMSprop pentru clasele: ”Shoes”, ”Flip Flops” și ”Sandals”

Urmatorul algoritmul pentru clasificarea îmbrăcăminți operează cu trei clase distincte: ”Topwear”, ”Bottomwear”, ”Dress”. Observăm că optimizatorul Adam a obținut performanțe superioare pe un număr mai mic de epoci, în timp ce RMSprop a demonstrat performanțe mai bune decât Adam pe un număr mai mare de epoci.

Această diferență poate fi atribuită, în parte, adaptabilității diferite a celor două optimizatori la ratele de învățare variabile și la structurile complexe ale modelelor. Adam este recunoscut pentru adaptabilitatea sa și pentru capacitatea de a convergența rapid către minimele locale, ceea ce îi conferă un avantaj în fazele inițiale ale antrenamentului. Pe de altă parte, RMSprop poate oferi o direcție de căutare mai precisă a minimului local în etapele finale ale antrenamentului, ceea ce îi permite să performeze mai bine pe termen lung. Astfel, în timp ce Adam se remarcă prin capacitatea sa de a obține rezultate bune în etapele incipiente, RMSprop arată o convergență mai stabilă și performanțe superioare pe termen lung.

Epochs	Adam		RMSprop	
	Antrenare	Validare	Antrenare	Validare
10 epoci	75%	73%	65%	65%
80 epoci	82%	81%	83%	80%

Tabela 3.4: Comparație între optimizatorii Adam și RMSprop pentru clasele: ”Topwear”, ”Bottomwear”, ”Dress”

ArticleType În cadrul acestei categorii s-au dezvoltat trei algoritmi pentru clasificarea articolelor de îmbrăcăminte, unul pentru articole de încălțăminte și unul pentru diferențierea tricourilor.

Epochs	Adam		RMSprop	
	Antrenare	Validare	Antrenare	Validare
10 epoci	80%	79%	80%	77%
80 epoci	80%	80%	80%	79%

Tabela 3.5: Comparație între optimizatorii Adam și RMSprop pentru rezutatele clasifcării îmbrăcăminții

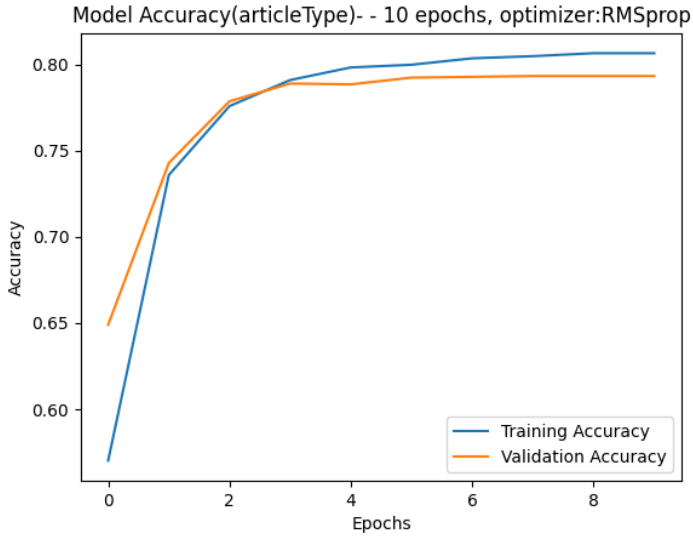

Figura 3.5: Acuratetea modelului pe 10 epoci cu optimizator Adam

Se observă că, pentru ambele seturi de epoci (10 și 80), precizia modelului în timpul antrenării și validării este relativ constantă și se situează în jurul valorii de 80%. Acest lucru sugerează că modelul are o performanță stabilă și nu prezintă o variație semnificativă între seturile de epoci evaluate.

De asemenea, se poate observa că nu există o diferență semnificativă între precizia modelului în timpul antrenării și validării, indicând faptul că modelul nu suferă de overfitting sau underfitting. Rezultatele constante între cele două seturi de epoci sugerează o bună generalizare a modelului la datele nevăzute și că antrenamentul suplimentar nu aduce îmbunătățiri semnificative.

Epochs	**Adam**		**RMSprop**	
	Antrenare	**Validare**	**Antrenare**	**Validare**
10 epoci	36%	30%	42%	41 %
80 epoci	42%	41%	42%	41%

Tabela 3.6: Comparație între optimizatorii Adam și RMSprop pentru rezutatele clasifcării încălțăminții

Tabelul 3.6 reflectă performanța modelului, evidențiind rezultatele în timpul antrenării și validării pentru două seturi de epoci, 10 și 80, utilizând modelul pre-antrenat VGG16. Deși s-au încercat diferite abordări, inclusiv utilizarea modelelor pre-antrenate și implementarea unui model personalizat.

În particular, chiar și cu utilizarea modelului pre-antrenat VGG16, observăm că precizia nu atinge nivelurile dorite. Acest lucru poate sugera că datele în sine pot fi dificile de învățat sau că poate fi necesară o ajustare suplimentară a parametrilor modelului.

Epochs	**Adam**		**RMSprop**	
	Antrenare	**Validare**	**Antrenare**	**Validare**
10 epoci	58%	56%	59%	57 %
80 epoci	72%	70%	77%	76%

Tabela 3.7: Comparație între optimizatorii Adam și RMSprop pentru rezutatele clasifcării între: ”Tshirts”, ”Shirts”, ”Tops”

Cât despre diferețierea dintre tricouri, bluze și topuri algoritmul are rezultate relativ bune.Observăm că, pe măsură ce numărul de epoci crește, performanța modelului în creștere atât pe setul de date de antrenare, cât și pe cel de validare. La 10 epoci, acuratețea pe setul de date de antrenare este în jur de 58%, în timp ce pe setul de date de validare este de aproximativ 56%. Aceste cifre cresc la 80 de epoci, atingând aproximativ 72% pentru antrenare și 70% pentru validare.

Style Cum se poate observa în tabela 3.8 pentru ambii optimizatori, atât pentru 10 epoci, cât și pentru 80 de epoci, precizia este consistentă și destul de înaltă, variază între 83% și 84% pentru setul de date de validare și între 84% și 84% pentru setul de date de antrenament. Această consistență indică faptul că modelul are capacitatea de a învăța și generaliza bine în timpul antrenamentului și că nu există semne clare de overfitting.

Epochs	**Adam**		**RMSprop**	
	Antrenare	**Validare**	**Antrenare**	**Validare**
10 epoci	84%	84%	84%	83%
80 epoci	84%	84%	84%	84%

Tabela 3.8: Comparație între optimizatorii Adam și RMSprop pentru rezutatele clasifcării stilului

Seson În procesul de antrenare, am experimentat cu trei modele diferite: VGG, MobileNetV2 și SingleOutputModel. Am încercat să evaluăm performanța acestor modele pe setul nostru de date pentru clasificarea imaginilor. După mai multe iterații de antrenare și ajustări ale hiperparametrilor, am constatat că cele mai bune rezultate au fost obținute folosind arhitectura MobileNetV2.

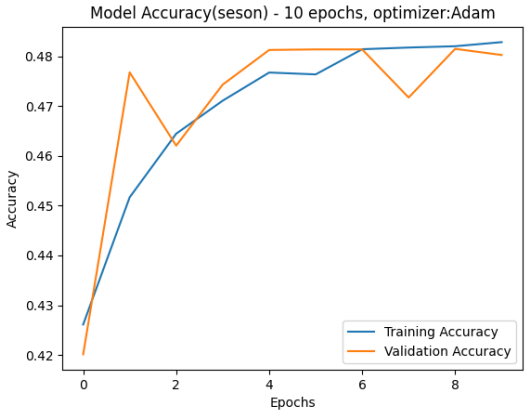

Figura 3.6: Acuratetea modelului pe 10 epoci cu optimizator Adam

Observăm că, pentru ambele seturi de epoci, acuratețea pe setul de antrenare și pe cel de validare este relativ constantă, în jurul valorii de 48% pentru 10 epoci și 48% pentru 80 epoci. Acest lucru sugerează că modelul nu se îmbunătățește semnificativ

în timpul procesului de antrenare extins.În plus, se poate observa că performanța pe setul de validare este în general puțin mai mică decât pe setul de antrenare, indicând o posibilitatea ca modelul să facă overfitting.

Epochs	**Adam**		**RMSprop**	
	Antrenare	**Validare**	**Antrenare**	**Validare**
10 epoci	48%	48%	40%	39%
80 epoci	48%	48%	48%	46%

Tabela 3.9: Comparație între optimizatorii Adam și RMSprop pentru rezutatele clasifcării sezonului

Concluzii

Performanța modelelor de învățare automată este un aspect crucial în domeniul inteligenței artificiale, iar îmbunătățirea acesteia necesită o abordare atentă și conștientă a mai multor factori interdependenți. Este evident că calitatea și cantitatea datelor disponibile joacă un rol crucial în performanța modelelor de învățare automată. În cazul de față, o extindere și diversificare a setului de date ar putea contribui semnificativ la îmbunătățirea acurateței și a generalizării modelului. Un set de date mai mare și mai diversificat ar permite modelului să învețe și să recunoască o gamă mai largă de articole de îmbrăcăminte, ceea ce ar conduce la recomandări mai precise și mai variate.

În plus, soluția propusă ar putea fi integrată cu succes într-o aplicație care oferă recomandări de outfituri. Prin implementarea modelului într-o astfel de aplicație, utilizatorii ar putea beneficia de sfaturi personalizate și inspirație în alegerea ținutelor potrivite pentru diverse ocazii. Integrarea soluției într-o aplicație ar aduce un plus de valoare și ar face accesibilă tehnologia de clasificare a articolelor de îmbrăcăminte unui număr mai mare de utilizatori.

În final, îmbunătățirea performanței modelelor de învățare automată necesită o abordare continuă, iterativă și adaptabilă. Prin explorarea constantă a diferitelor tehnici și abordări, și prin adaptarea lor la specificul problemei și datelor disponibile, se poate obține o performanță mai bună

Bibliografie

[Keras] Biblioteca Keras. Disponibilă la: https://keras.io/

[TensorFlow] Biblioteca OpenCV. Disponibilă la: https://www.tensorflow.org/api_docs

[Opencv] Biblioteca OpenCV. Disponibilă la: https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html

[Pandas] Biblioteca Pandas. Disponibilă la: https://pandas.pydata.org

[Numpy] Biblioteca Numpy. Disponibilă la: https://numpy.org

[Mtplotlib] Biblioteca Matplotlib. Disponibilă la: https://matplotlib.org

[Scikit-learn] Biblioteca Scikit-learn. Disponibilă la: https://scikit-learn.org/stable/

[Kaggle] Kaggle. Disponibil la: https://www.kaggle.com.

[FashionDataset] Kaggle - Fashion Product Images (Small). Disponibil la: https://www.kaggle.com/datasets/paramaggarwal/fashion-product-images-small.