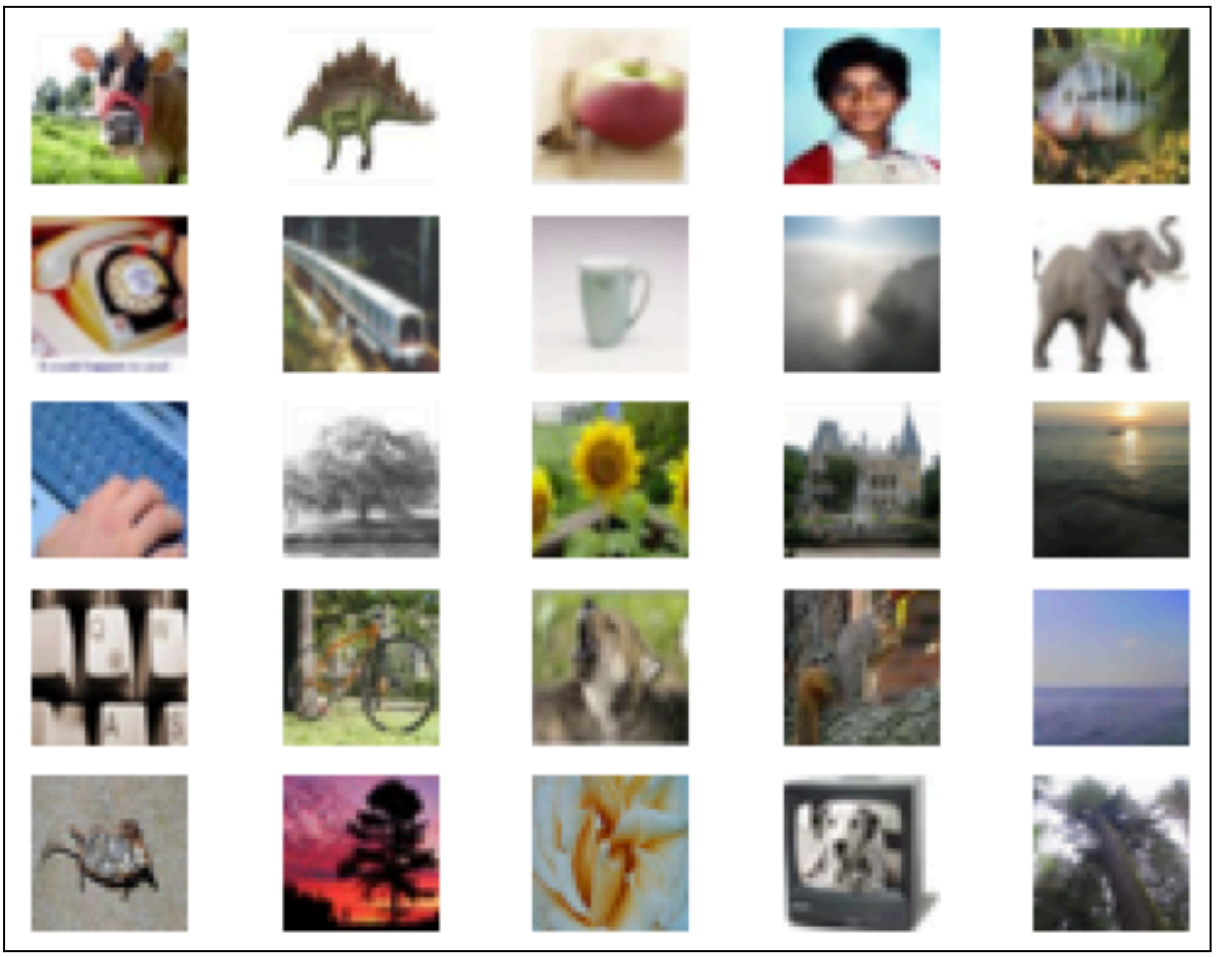
Sztuczne Sieci Neuronowe
Projekt

Cel projektu:
Stworzenie kompleksowego projektu, który obejmuje zarówno zadanie klasyfikacji obrazów (zbiór CIFAR-100) jak i zadanie regresji (zbiór California housing price).

Etapy projektu:
1. Zrozumienie danych - Przeanalizuj strukturę i charakterystyki zbioru CIFAR-100 (klasyfikacja obrazów) oraz California housing price dataset (regresja).
2. Przygotowanie danych
3. Opracowanie modeli sieci neuronowych (płytkich i głębokich) do klasyfikacji i regresji
Bazując na wiedzy uzyskanej w trakcie zajęć należy zaproponować różne modele sieci neuronowych do klasyfikacji i regresji. Odpowiednio je nauczyć oraz zoptymalizować ich działanie m.in. przez dobór odpowiednich hiperparametrów.
4. Ocena opracowanych modeli
Dokonać oceny analizowanych modeli wykorzystując odpowiednie miary. Można wykorzystać narzędzia do wizualizacji wyników, takie jak krzywe uczenia, macierze pomyłek itp.
5. Porównanie skuteczności otrzymanych modeli.
Przeanalizuj uzyskane wyniki.
Omów, jakie wyzwania pojawiły się w trakcie pracy nad projektem.

1. **CIFAR - 100**

Zbiór Cifar-100 składa się z obrazów podzielonych na 100 klas (ryba, kwiat, ssak itp.). Każda klasa zawiera 600 przykładowych grafik. Przykładowe grafiki zostały zaprezentowane poniżej:



Ze względu na ilość klas,, zmienna Y została podzielona na 100 kategorii:

```
y_train, y_test = to_categorical(y_train, number_of_classes),
                  to_categorical(y_test, number_of_classes)
```

W trakcie wykonywania projektu zostało wykonanych wiele eksperymentów tworzenia modeli wykorzystując sieci konwolucyjne oraz różne modele pretrenowane. W pliku jupyter notebook zostały zapisane modele dające najwyższą jakość klasyfikacji. Pozostałe eksperymenty zostały pominięte w celu utrzymania przejrzystości pliku.

a. Sieci Konwolucyjne

Eksperymenty wykonywane przy sieciach konwolucyjnych różniły się znacząco w kwestii struktury nauczanego modelu. Zmieniane były poniższe czynniki:

- ilość warstw
    - Zbyt mała ilość warstw nie pozwalała na odpowiednio wysokie wyuczenie się modelu, natomiast zbyt duża ich ilość prowadziła do przeuczenia modelu (overfitting) - wyniki na danych uczących były znacząco większe niż wyniki otrzymane na zbiorze testowym.
- typy warstw (Pooling, Dropout, Flatten, Conv2D, Dense)
    - Dodatkowe warstwy były dodawane w celu lepszego dostosowania modelu, zapobiegnięciu przeuczeniu (Dropout) oraz zapobiegnięcia zbyt długiego czasu nauczania (Pooling).
- Parametry poszczególnych warstw, rozmiar i struktura danych, funkcja aktywacji, padding)
    - W celu zapewnienia odpowiedniej ‘głębokości’ uczenia został zastosowany padding. Struktura danych została dopasowana do zbioru Cifar-100 zawierającego 100 klas
- ilość iteracji
    - Ilość iteracji była dostosowywana do skomplikowania sieci. Przy prostej strukturze zbyt duża ilość iteracji miała negatywny wpływ na jakość klasyfikacji lub nie wpływała na wyniki, ponieważ modelu szybko dostosowywał się do danych uczących.
- wartość współczynnika uczenia
    - Zbyt duży współczynnik uczenia prowadził do wysokich skoków wartości wag w poszczególnych iteracjach, natomiast zbyt mały współczynnik sprawiał, że zmiany wartości wag były zbyt małe.
- Zastosowanie regulacji

Wykorzystane modele:

```
[ ]  model = Sequential([
         Conv2D(128, (3, 3), padding='same', input_shape=x_train.shape[1:], activation='elu'),
         MaxPooling2D(pool_size=(2,2)),
         Conv2D(128, (3, 3), activation='elu'),
         MaxPooling2D(pool_size=(2, 2)),
         Conv2D(256, (3, 3), padding='same', activation='elu'),
         MaxPooling2D(pool_size=(2, 2)),
         Dropout(0.25),
         Conv2D(512, (3, 3), padding='same', activation='elu'),
         MaxPooling2D(pool_size=(2, 2)),
         Dropout(0.25),
         Flatten(),
         Dense(1024, activation='elu'),
         Dropout(0.5),
         Dense(number_of_classes, activation='softmax'),
     ])

     model.compile(
         optimizer='adam',
         loss='categorical_crossentropy',
         metrics=['accuracy']
     )

     data_generator = ImageDataGenerator(
         featurewise_center=False,
         samplewise_center = False,
         featurewise_std_normalization=False,
         samplewise_std_normalization=False,
         zca_whitening=False,
         width_shift_range=0.1,
         height_shift_range=0.1,
         horizontal_flip=True,
         vertical_flip=False,
         rotation_range=0
     )

     data_generator.fit(x_train)

     early_stop = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=8, restore_best_weights=True)

     model.fit(
         data_generator.flow(x_train, y_train, batch_size=50),
         steps_per_epoch=x_train.shape[0]//50,
         epochs=25,
         validation_data=(x_test, y_test),
         verbose=1,
         callbacks=[early_stop]
         )
```

```
base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=(32, 32, 3))

base_model.trainable = True

model = Sequential([
    base_model,
    GlobalAveragePooling2D(),
    Dense(256, activation='relu'),
    BatchNormalization(),
    Dense(128, activation='relu'),
    BatchNormalization(),
    Dense(100, activation='softmax')
])

model.compile(optimizer=Adam(learning_rate=0.0001),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=30, validation_data=(x_test, y_test))
```

```
base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=(32, 32, 3))

for layer in base_model.layers[:-20]:
    layer.trainable = True

model = Sequential([
    base_model,
    GlobalAveragePooling2D(),
    Dense(256, activation='relu'),
    BatchNormalization(),
    Dense(128, activation='relu'),
    BatchNormalization(),
    Dense(100, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=30, validation_data=(x_test, y_test))
```

```
base_model = Xception(weights='imagenet', include_top=False, input_shape=(75, 75, 3))

base_model.trainable = True

model = Sequential([
    base_model,
    GlobalAveragePooling2D(),
    Dense(256, activation='relu'),
    BatchNormalization(),
    Dense(128, activation='relu'),
    BatchNormalization(),
    Dense(100, activation='softmax')
])

model.compile(optimizer=Adam(learning_rate=0.001),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train_resized, y_train, epochs=7, validation_data=(x_test_resized, y_test))
```

Jakości klasyfikacji testowanych modeli osiągały znacząco różne wyniki. Najsłabsze modele osiągały efektywność mniejszą niż 1%, najlepsze - powyżej 40%. Największy wpływ na jakość klasyfikacji miało dodanie różnych typów warstw, takich jak Pooling, Dropout, Flatten. Warstwy te odpowiadały za odpowiednie uproszczenie danych oraz dostosowanie modelu do tych danych. Podobny wpływ miało również dodanie większej ilości sekwencji - warstwa konwolucyjna i warstwa pooling.

Ostateczny, najbardziej skuteczny model osiągnął jakość klasyfikacji równą 44,86%. Przy tym modelu zostały wykorzystane dwie dodatkowe techniki - sztuczne zwiększenie danych (data augmentation) oraz zastosowanie przedwczesnego zatrzymania procesu nauki.

Data Augmentation odpowiada za sztuczne zwiększenie ilości danych poprzez takie operacje jak: rotacja, zwiększanie szerokości/wysokości obrazów. Technika ta umożliwiła przeprowadzenie nauczania z wykorzystaniem większej ilości iteracji bez narażania modelu na zbyt szybkie dopasowanie się do danych uczonych.

Warunek stop został użyty w celu zatrzymania procesu nauki w sytuacji, kiedy błąd zaczął wzrastać. Funkcja *EarlyStopping* umożliwiła również zachowanie optymalnych wag w przypadku otrzymania gorszych wartości w kolejnych iteracjach procesu nauczania

W przypadku modeli pretrenowanych testowane były takie same parametry, jak przy sieciach konwolucyjnych. Dodatkowo eksperymenty były również przeprowadzane z użyciem różnych modeli pretrenowanych (MobileNetV2, Xception, InceptionV3, EfficientNetV0) oraz fine-tuning'em danych. Jakości klasyfikacji wahały się od wartości mniejszych niż 1% do prawie 70%.

W przypadku danych Cifar-100 najskuteczniejszymi modelami okazały się MobileNetV2 (54,49%) oraz Xception (68,07%). Najlepszy wynik wyniósł 68,07% z użyciem modelu pretrenowanego Xception. W tym przypadku Fine-Tuning został zastosowany na wszystkich warstwach. pozwalało to na dodatkową poprawę wcześniej wyuczonych warstw modelu pretrenowanego.

Podsumowując, najlepszą jakość klasyfikacji danych Cifar-100 osiągnęła sieć wykorzystująca model pretrenowany Xception oraz Fine-Tuning. W tym wypadku jakość klasyfikacji osiągnęła 68,07%. Modele wykorzystujące transfer-learning dawały znacząco bardziej satysfakcjonujące wyniki niż sieci konwolucyjne budowane od podstaw

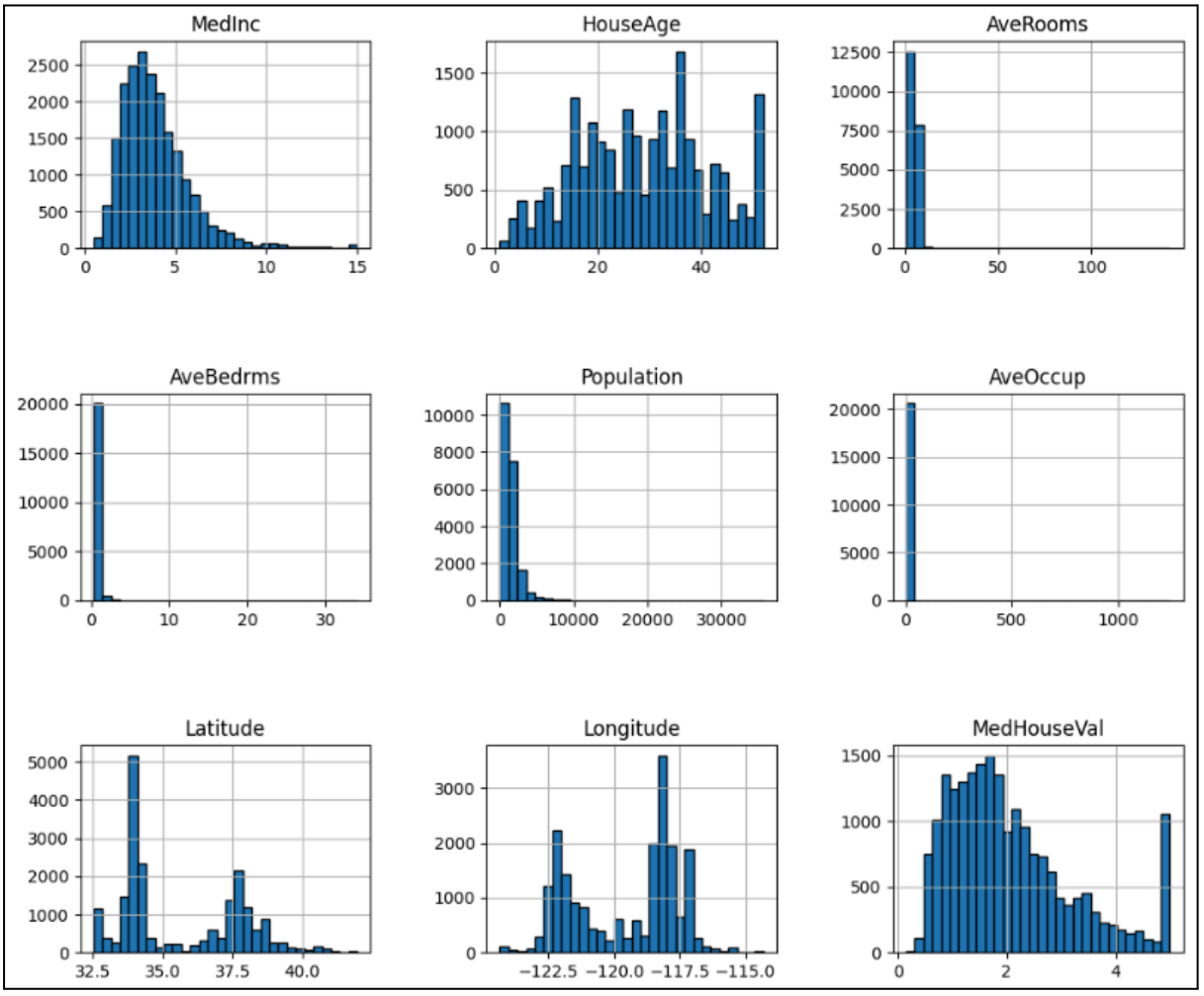
2. California Housing

Zbiór California Housing składa się z 8 zmiennych niezależnych oraz zmiennej zależnej, która wyznacza średnią wartość domów w danej okolicy (w $100 000). Do zmiennych niezależnych należą:

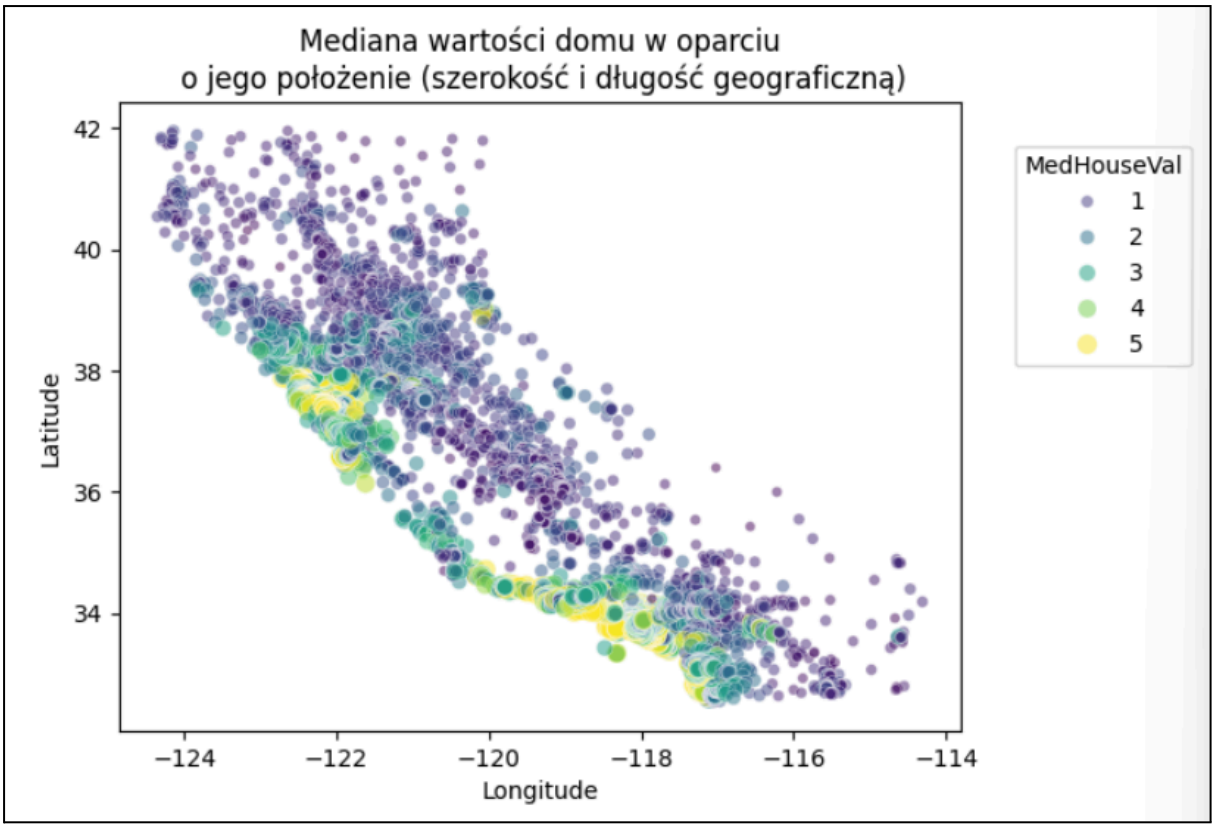
- *MedInc* - średni dochód pobliskich mieszkańców
- *HouseAge* - średni wiek domów w pobliżu
- *AveRooms* - średnia ilość pokoi w pobliskich domach
- *AveBedrms* - średnia ilość sypialni w pobliskich domach
- *Population* - ilość mieszkańców w pobliżu
- *AveOccup* - średnia ilość mieszkańców w poszczególnych domach
- *Latitude* - szerokość geograficzna
- *Longitude* - długość geograficzna

Zbiór danych California Housing pozwalał na dużo dokładniejszą wstępną analizę danych, ze względu na liczbowe wartości danych. W celu bardziej dogłębnego zrozumienia zbioru zostało przedstawione kilka wykresów pokazujących zależności pomiędzy poszczególnymi atrybutami a ceną. Wizualizacje te pozwoliły na zauważenie kilku ważnych więzi.

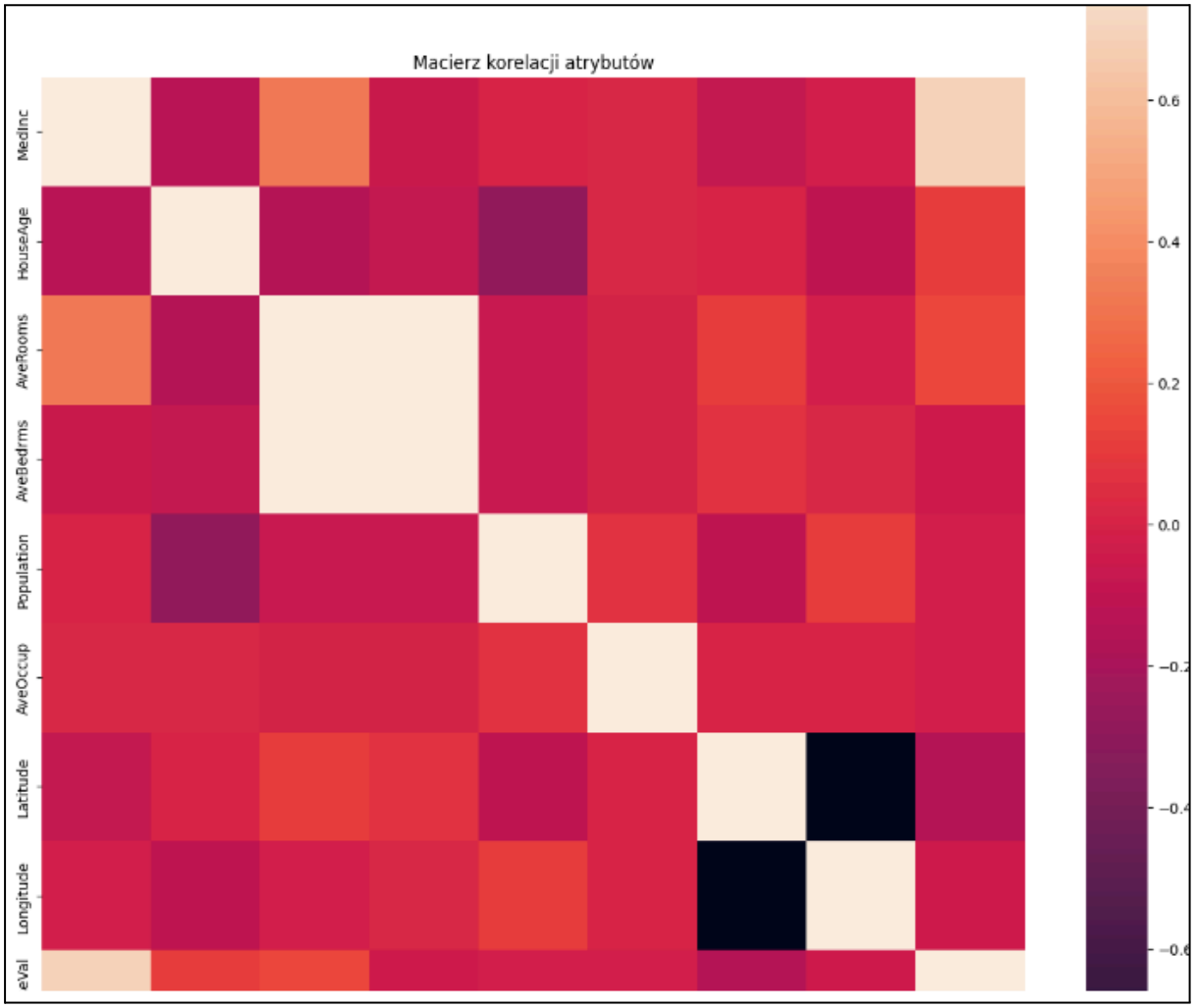
a. Histogramy - Na 9 histogramach została przedstawiona wartość domu w zależności od poszczególnych parametrów. Do istotnych obserwacji należą:
    - Znaczące powiązanie pomiędzy średnich dochodem mieszkańców w pobliżu domu do jego ceny
    - Znaczny wzrost ceny dla poszczególnych wartości długości i szerokości geograficznej
    - Znaczący wzrost ceny domu przy większej ilości mieszkańców w pobliżu



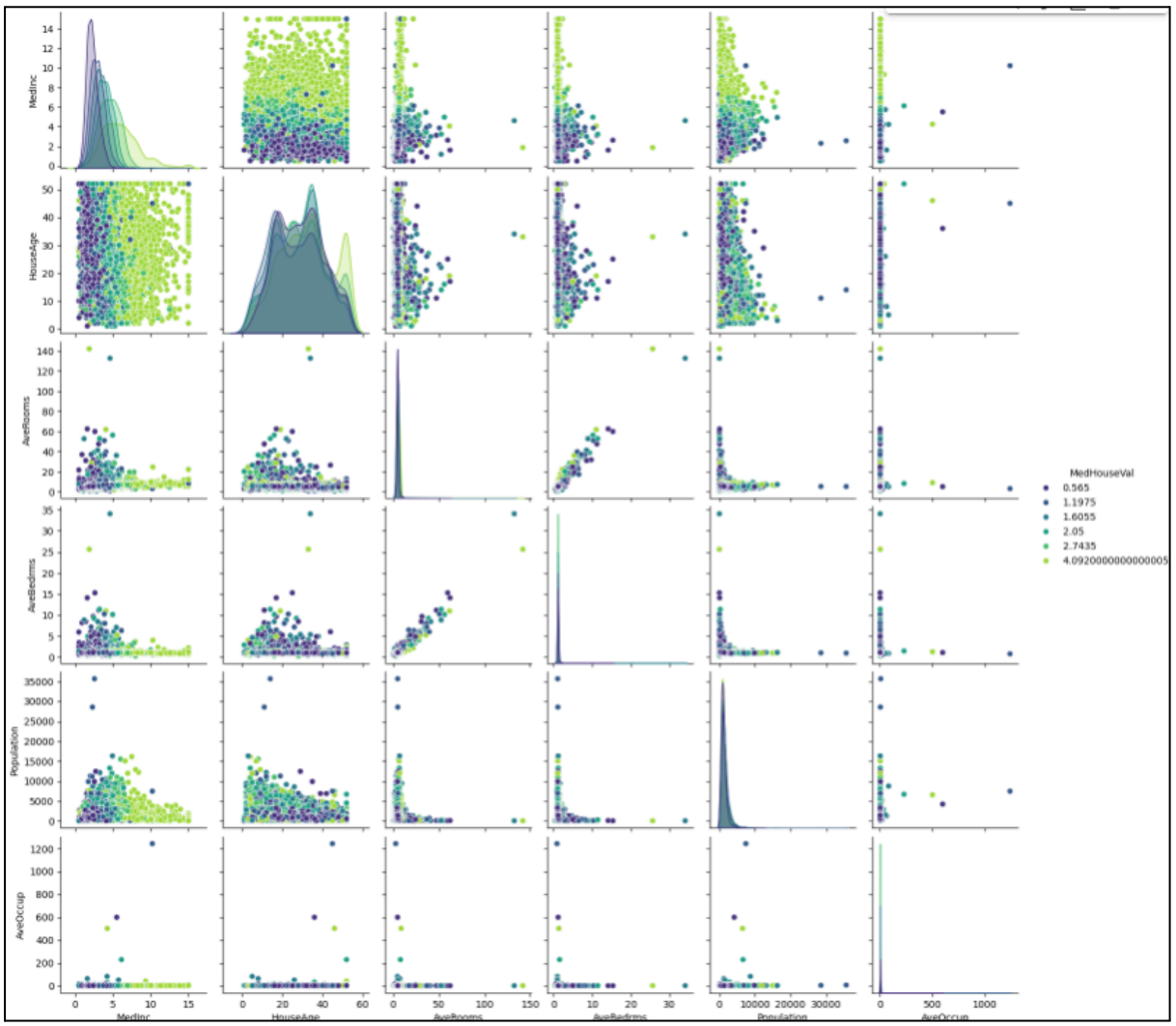
b. Wykres punktowy - Wykres ten przedstawia uzależnienie położenia domów od średniej ceny. Można zauważyć, że domy o podobnych wartościach mają bliskie sobie położenie, co oznacza, że długość i szerokość geograficzna mają znaczący wpływ na cenę.



c. Macierz korelacji - Macierz przedstawia skorelowanie poszczególnych atrybutów z ceną. Po raz kolejny zauważalna jest ogromna zależność pomiędzy położeniem domu, a jego ceną.



d. Macierzowe wykresy rozrzutu - wykresy te przedstawiają zależności pomiędzy poszczególnymi atrybutami. Na przekątnej znajdują się histogramy poszczególnych zmiennych.



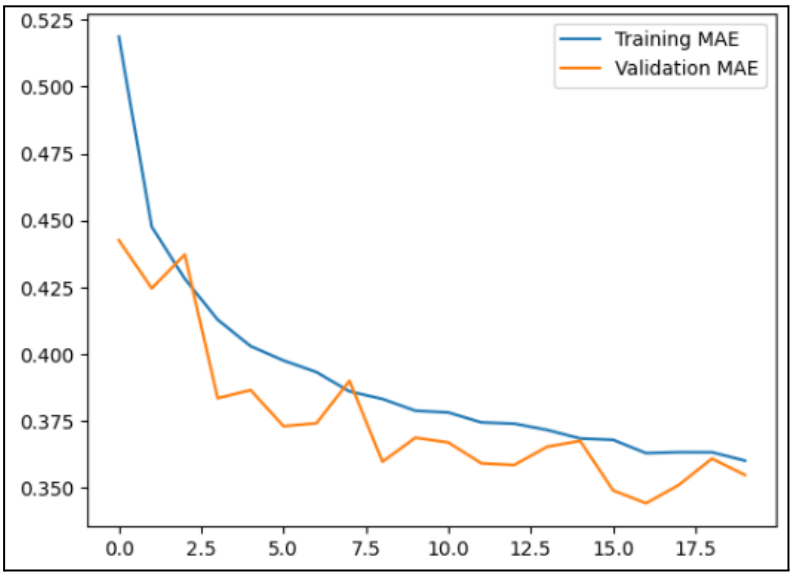
Dodatkowo, została przeprowadzona również normalizacja danych. Technika ta pozwala na dokładniejsze nauczanie modelu, poprzez zmniejszenie różnic wartości pomiędzy poszczególnymi parametrami.

Przeprowadzono badania na modelach płytkich o rozmiarach 256x256x128 128x128x128 64x256x128.

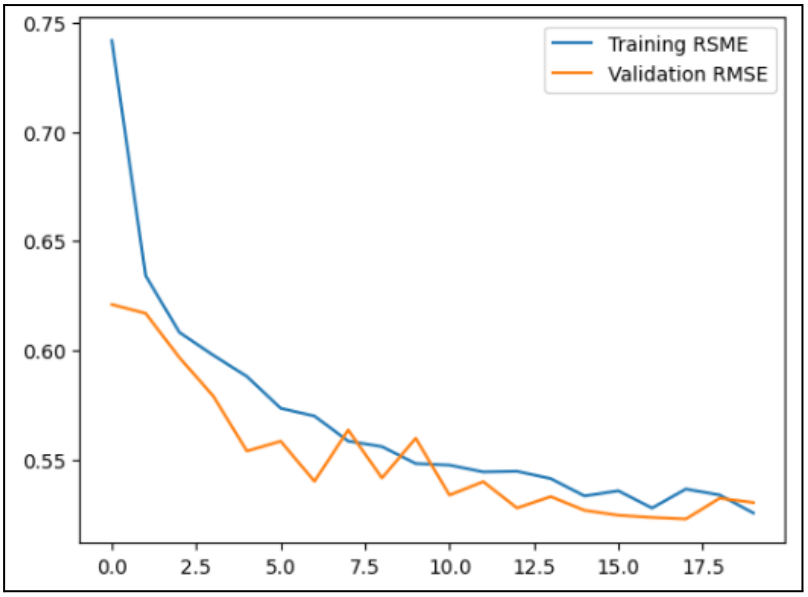
```
model = Sequential([
    Dense(256, activation='relu'),
    Dropout(0.2),
    Dense(256, activation='relu'),
    Dropout(0.2),
    Dense(128, activation='relu'),
    Dense(1, activation='linear')
])

rmse = tf.keras.metrics.RootMeanSquaredError()

model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mean_absolute_error', rmse])

history = model.fit(
    x_california_train,
    y_california_train,
    validation_data=(x_california_test, y_california_test),
    epochs = 20, batch_size=16)
```

W celu testowania efektywności modelu wykorzystane zostały:

- średni błąd bezwzględny



- błąd średniokwadratowy
- pierwiastek błędu średniokwadratowego



- wariancja (rozproszenie) wyników

Ostatecznie, najwyższą jakość klasyfikacji otrzymał mało skomplikowany model z liniową funkcją aktywacji w warstwie wyjściowej o rozmiarach 256x256x128. Wyniki otrzymane dla tego modelu przedstawione są poniżej:

- średni błąd bezwzględny: `0.3547901520672579`
- błąd średniokwadratowy: `0.281291878647357`
- pierwiastek błędu średniokwadratowego: `0.5303695679876034`
- wariancja (rozproszenie) wyników: `0.7897725121876026`